НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Веб-платформа з продажу та обміну токенізованих активів*

Виконав: студент IV курсу, групи *ІП-61 Кролевецький Денис Юрійович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. викладач, Олійник Ю.О.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

Доц., каф ТК, к.т.н., доц. Ткач М.М.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6113. 045440.01.81	Пояснювальна записка	55	
3	А4	КПІ.ІП-6113. 045440.02.13	Опис програми	14	
4	А4	КПІ.ІП-6113. 045440.03.91	Технічне завдання	12	
5	А4	КПІ.ІП-6113. 045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-6113. 045440.05.34	Керівництво користувача	8	
7	А3	КПІ.ІП-6113. 045440.06.99 СС	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-6113. 045440.06.99 СС	Схема структурна бізнес процесів	1	
9	А3	КПІ.ІП-6113. 045440.06.99 КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6113.045440.00.90		
Розробн.		Кролевецький Д.Ю.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Олійник Ю.О.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-61	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Кролевецькому Денису Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Веб-платформа з продажу та обміну токенізованих* активів»

керівник проєкту *Олійник Юрій Олександрович*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення

3) Аналіз якості та тестування програмного забезпечення

4) Впровадження та супровід програмного забезпечення

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна бізнес процесів

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*28.02.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*07.03.2020*	
3.	*Постановка та формалізація задачі*	*12.03. 2020*	
4.	*Аналіз вимог до програмного забезпечення*	*17.03. 2020*	
5.	*Алгоритмізація задачі*	*25.03. 2020*	
6.	*Моделювання програмного забезпечення*	*29.03. 2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*04.04. 2020*	
8.	*Розробка архітектури програмного забезпечення*	*19.04. 2020*	
9.	*Розробка програмного забезпечення*	*02.05. 2020*	
10.	*Налагодження програми*	*06.05. 2020*	
11.	*Виконання графічних документів*	*09.05.2020*	
12.	*Оформлення пояснювальної записки*	*22.05.2020*	
13.	*Подання ДП на попередній захист*	*29.05.2020*	
14.	*Подання ДП рецензенту*	*06.06.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Денис КРОЛЕВЕЦЬКИЙ
(підпис)

Керівник _____ Юрій ОЛІЙНИК
(підпис)

Пояснювальна записка
до дипломного проєкту

на тему: Веб-платформа з продажу та обміну токенізованих активів

Київ – 2020 року

АНОТАЦІЯ

Робота містить 16 рисунків і 16 таблиць.

Технологія блокчейн отримує все більше розповсюдження з кожним роком. Починають з’являтися нові підходи та продукти, які змінюють уявлення про те, як можуть працювати ті чи інші сервіси. Технологія починає використовуватись більш широко у різних сферах розробки.

Одна з найбільших переваг технології є децентралізація. Мається на увазі, що сервіс не залежіть від певного провайдера чи сервера (серверів), на яких зберігаються данні. Правильна архітектура мережі блокчейн та продукту дозволяє уникнути втрати даних чи несанкціонованих змін. Таким чином користувач відчуває довіру до сервісу та здатен самостійно переконатися у правильності його роботи.

Токенізація – один із прикладів впровадження технології блокчейн у існуючі рішення. Вона передбачає виділення певної сутності з продукту та переміщення логіки взаємодії з нею у мережу Blockchain. Такий підхід дозволяє користувачу взаємодіяти з власними активами без прямої залежності від певного серверу чи інших користувачів.

У першому розділі було розглянуто підходи то токенізації, та обрано кращий для впровадження.

У другому розділі було описано архітектуру програмного продукту, який являє собою сервіс з продажу книг, які були токенізовані. Логіка взаємодії з ними була переміщена у мережу Ethereum.

У третьому розділі було проаналізовано якість програмного забезпечення, описано методику тестування та продемонстровано приклади тестів.

У четвертому розділі було описано розгортання програмного забезпечення, а також детально розглянуто роботу з сервісом.

ABSTRACT

The work contains 16 figures and 16 tables.

Blockchain technology is becoming more widespread every year. New approaches and products are beginning to emerge that are changing the way certain services can work. The technology is beginning to be used more widely in various fields of development.

One of the biggest advantages of technology is decentralization. This means that the service does not depend on the particular provider or server (s) on which the data is stored. Proper blockchain and product network architecture avoid data loss or unauthorized changes. Thus, the user feels confident in the service and is able to see for themselves the correctness of its work.

Tokenization is one example of the introduction of blockchain technology into existing solutions. It involves extracting a certain entity from the product and moving the logic of interaction with it to the Blockchain network. This approach allows the user to interact with their own assets without directly depending on a particular server or other users.

In the first section, approaches to tokenization were considered, and the best one for implementation was selected.

The second section described the architecture of the software product, which is a service for selling books that have been tokenized. The logic of interacting with them has been moved to the Ethereum network.

The third section analyzed the quality of the software, described the testing methodology, and demonstrated examples of tests.

The fourth section described software deployment, as well as detailed work with the service.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface

REST – Representational State Transfer

Ethereum – Одна з мереж, побудована з використанням технології Blockchain

POW – Протокол (правило) створення нових блоків у мережах Blockchain

ETH – Основний токен мережі Ethereum з однойменною назвою

P2P – Пряма взаємодія між користувачами певного продукту

Нода – учасник (частина) блокчейн мережі

ID – ідентифікатор

ВСТУП

Технологія блокчейн отримала значний розвиток за останні роки. З’явилось багато різних мереж та протоколів їх роботи, а разом із ними – нові підходи до створення сервісів. Набула поширення концепція DAPP – децентралізований додаток у якому відсутній центральний чи головний сервіс. Користувачі взаємодіють на рівних умовах слідуючи чітко прописаним правилам у смарт контрактах. Смарт контракти являють собою пов’язані між собою класи та методи, які записані у мережі Blockchain, для виконання тих чи інших математичних операцій чи роботи з даними.

Невід'ємною частиною концепції DAPP є смарт контракти токенів. Смарт контракт токену в даному контексті означає запис у мережі блокчейн про стан рахунку умовних одиниць певного. Також сюди відносять функції керування та зміни станів цих рахунків. Токени дають можливість певним чином провести відцифрування реальних активів та отримати значні інструменти управління для користувача ними без посередника у вигляді певного сервісу.

Актуальність теми – токенізація активів, як підхід до відцифрування, дозволяє отримати значні інструменти керування власними активами не маючи при цьому залежність від того чи іншого сервісу. Такий підхід дозволяє підвищити прозорість роботи продуктів та надає користувачам переконатись у правильності та чесності їх роботи.

Мета – збільшення можливостей з продажу та обміну цифровими активами.

Призначення – інформаційна підтримка процесу продажу та обміну цифровими активами.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Блокчейн – побудований за визначеними правилами ланцюг блоків, які містять інформацію. Цілісність ланцюга забезпечується тим, що кожен блок містить у собі як власну хеш-суму, так і хеш-суму попередніх блоків.

Протокол мережі блокчейн – набір правил за якими визначається учасник мережі, який створить запис про новий блок та отримає винагородження. У контексті даної роботи будемо розглядати мережі на основі протоколу POW (Proof of Work). Основна ідея даного протоколу полягає у тому, що найбільший шанс створити новий блок має той учасник мережі, у якого найбільші розрахункові потужності.

Транзакція – операція зміни стану мережі блокчейн. Транзакції групуються у блоки і таким чином потрапляють у мережу.

Смарт контракт – запис у мережі блокчейн який являє собою структурований набір даних та інструкцій (функцій) зміни цих даних. З точки зору програмного коду – являє собою пов’язаний набір класів з функціями та даними. Усі дані в рамках смарт контракту є публічними, але їх зміна обмежена закладеною логікою.

Смарт контракт токену – різновид смарт контрактів які складаються з рахунків користувачів, та функцій зміни станів цих рахунків. Токен – умовна одиниця рахунку користувача.

1.2 Змістовний опис і аналіз предметної області

Існують різні підходи до реалізації смарт контрактів токенів. Виділяють стандарти токенів – певні домовленості, щодо інтерфейсів та реалізацій. Особливості кожного стандарту роблять його більш ефективним рішенням для певного класу задач.

1.2.1 Стандарт ERC-179

Найпростіший стандарт смарт контракту токену. З точки зору даних містить лише однозв’язний список, який вказує на те, скільки токенів належить певному гаманцю.

Функції:

– balanceOf – повертає баланс вказаного гаманця;

– transfer – переміщує вказану кількість токенів з гаманця відправника транзакції на вказаний гаманець. Відправник не може надіслати більше токенів ніж у нього є і, відповідно, має доступ лише до власного гаманця;

– mint – створює вказану кількість токенів на рахунку вказаного гаманець. Зазвичай відправник обов’язково має бути власником (той, хто створив) контракту.

Даний стандарт підходить для реалізації простих сервісів.

1.2.2 Стандарт ERC-20

Розширений стандарт ERC-179. Окрім функцій, властивих попередньому стандарту містить нові:

– approve – дозволяє вказаному гаманцю користуватися певною кількістю токенів на рахунку відправника;

– getApproved – повертає кількість токенів, якою може користуватися вказаний користувач з вказаного рахунку;

– transferFrom – переміщує вказану кількість токенів з вказаного рахунку на інший рахунок. Попередньо необхідно отримати дозвіл (функція approve).

Завдяки додатковим функціям з’являється можливість інтегрувати даний смарт контракт з іншими. Надається доступ іншим користувачам або контрактам на керування рахунками даного. Таким чином з’являється можливість створювати функції, які пов’язують декілька контрактів і користувач, наприклад, може обміняти одні токени на інші.

1.2.3 Стандарт ERC-721

За набором функцій даний стандарт дуже схожий на ERC-20. Відмінність полягає у самій сутності токенів.

У стандарті ERC-20 токен є взаємно рівним з усіма іншими. Тут можна провести аналогію з грошима – кожна банкнота відповідного номіналу вважається абсолютно рівною іншій банкноті того ж номіналу. Таким чином для у контексті контрактів ERC-20 важлива лише загальна кількість токенів на тому чи іншому гаманці.

На відміну від попереднього, ERC-721 має інший підхід. Токени даного стандарту створюються згідно певної моделі – фіксований набір параметрів (аналог struct). Але кожний окремий токен може мати різні значення цих параметрів, що дозволяє йому бути унікальним.

Як наслідок унікальності стандарт має дещо змінені функції – замість кількості токенів для маніпуляції вказується ідентифікатор певного токена.

1.3 Аналіз успішних ІТ-проєктів

1.3.1 Аналіз відомих технічних рішень

Проаналізувавши розповсюджені рішення можна зробити висновок, що різні проєкти використовують як підходи з токенами на великих блокчейн мережах так і створюють власні мережі. Більшість смарт контрактів токенів є у відкритому доступі, чого не можна зазначити про програмний код створених блокчейн мереж.

Програмний продукт, створений у рамках даної дипломної роботи використовує розповсюджені стандарти токенів та побудований на одній із великих мереж – Ethereum [1].

Таблиця 1.1 – Видомі технічні рішення

Тип	Назва	Переваги	Недоліки
Набір інструментів токенізації	TokenD	– Інструменти адміністрування – Підключення до фіатних платіжних сервісів + KYC – Маркетингові та біржові інструменти	– Токен - переважно внутрішня сутність – Переважно централізовані рішення – Необхідність підтримки
Токенізація фіатних активів	Крипто-гривня	– Легалізація регулятором (Нац. Банк України) – Гарантована вартість	– Повільні транзакції – Відносно висока комісія – Відсутні нструменти P2P обміну - лише купівля продаж у регулятора.

Продовження таблиці 1.1

Тип	Назва	Переваги	Недоліки
Державні сервіси	Земельний кадастр України	– Можливість перевірки даних – Легалізація регулятором (Держава Україна)	– Відносно вразливий блокчейн – Продукт не доведено до кінця (наявність даних у смарт контракті юридично не є повноцінним документом)

1.3.2 Аналіз відомих програмних продуктів

Виділимо наступні програмні продукти:

TokenD

Проєкт представляє набір інструментів та проведення токенізації певного сервісу [2].

Має багато різних інструментів для адміністрування продукту. Існує можливість підключення платіжних сервісів та проходження процедури верифікації особи. Переважно націлений на реалізацію маркетингового та біржового функціоналу. Дозволяє успішно пройти процедуру створення власної біржі для токенів або інтегруватися з існуючою.

До недоліків можна віднести те, що токен при використанні даного рішення є переважно особливістю внутрішньої архітектури а не самостійною одиницею. Побудоване рішення виходить централізованим, а отже не використовує архітектурні можливості мереж блокчейн у повному обсязі. Також рішення розповсюджується за платною підпискою та потребує постійно її підтримки.

Крипто-гривня

Приклад токенізованого рішення у банківській сфері.

Проєкт був запущений у розробку у 2016 [3] та має перші прототипи. Після завершення розробки матиме повну легалізацію регулятором (Національний Банк України). Через глибоку інтеграцію з фінансовими інститутами має токен цього проєкту має гарантовану вартість (1 токен = 1 гривня). Однією з особливостей, які з’являються завдяки впровадженню токенізації – можливість публічного відслідковування емісії.

До недоліків відноситься відносно низька швидкість транзакцій (значно нижча за сучасні банківські системи) та відсутність можливостей щодо інтеграції з іншими рішеннями для реалізації P2P обміну за інші токени. Рішення має місце лише у банківській сфері та не масштабується.

Земельний кадастр України

Приклад токенізації з використанням власної мережі блокчейн [4].

До переваг можна віднести легалізацію Державою Україна та визнання на законодавчому рівні.

До недоліків можна віднести слабку децентралізацію. Кількість нод обмежена та всі з них знаходяться під управлінням відповідного міністерства. Таким чином можна зробити висновок, що архітектура даної мережі блокчейн не надійна та з’являється можливість для маніпуляції даними. Блокчейн мережа у даній розробці не вирішує жодної архітектурної задачі чи задачі безпеки даних.

1.4 Аналіз вимог до програмного забезпечення

Для створення WEB-додатку за продажу токенізованих активів перш за все необхідно визначити ролі учасників. Клієнт – користувач, який може купувати, користуватись та обмінювати свої книги на інші. Адміністратор – користувач, який маже можливість додавати книги до каталогу або редагувати існуючі книги.

Даний програмний продукт повинен реалізовувати наступні можливості для клієнта:

– відображення доступних для купівлі книжок;

– можливість здійснення покупки;

– особистий кабінет;

– відображення доступних книжок для обміну;

– можливість здійснення обміну;

– можливість створення власного запиту на обмін.

Даний програмний продукт повинен реалізовувати наступні можливості для адміністратора:

– можливість додавання нових книжок;

– можливість редагування інформації про існуючі книги.

Детально розібрані зв’язки між акторами та функціональними модулями програмного продукту, варіанти використання додатка вказано в структурній схемі варіантів використання (Рисунок 1.1).

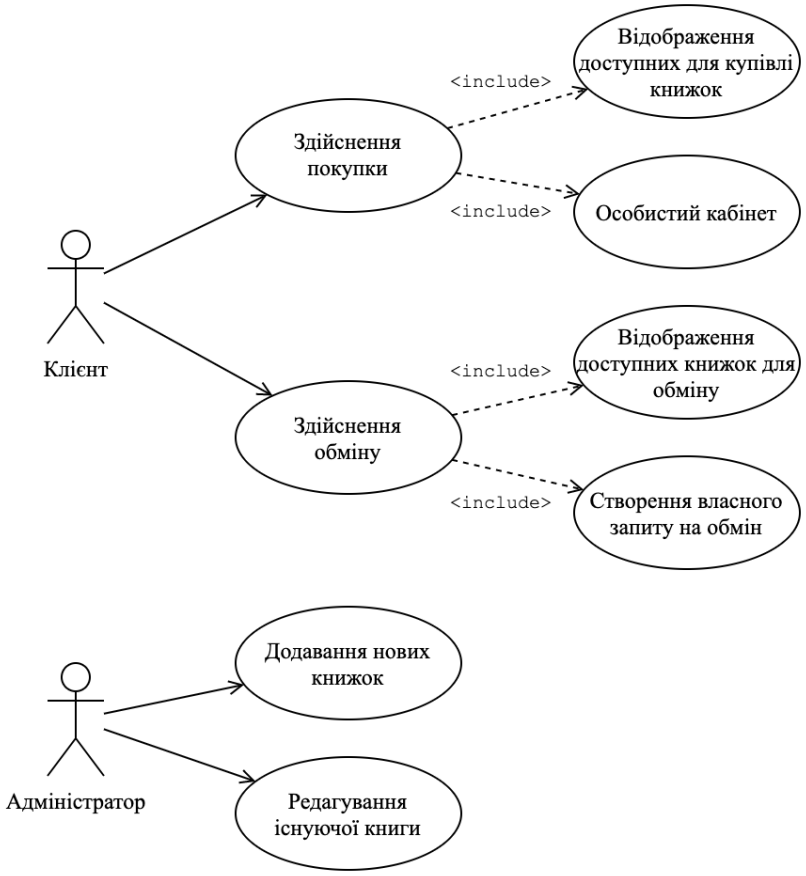


Рисунок 1.1 – Схема структурна варіантів використання

Матриця трасування вимог зображена на рисунку 1.2.

	Requirement: Можливість здійснення покупки	Requirement: Можливість здійснення обміну	Requirement: Можливість додавання нових книжок	Requirement: Можливість редагування інформації про існуючі книги
UseCase Model: Відображення доступних для купівлі книжок	+			
UseCase Model: Можливість здійснення покупки	+			
UseCase Model: Особистий кабінет	+			
UseCase Model: Відображення доступних книжок для обміну		+		
UseCase Model: Можливість здійснення обміну		+		
UseCase Model: Можливість створення власного запиту на обмін		+		
UseCase Model: Можливість додавання нових книжок			+	
UseCase Model: Можливість редагування інформації про існуючі книги				+

Рисунок 1.2 – Матриця трасування вимог

1.4.1 Розроблення функціональних вимог

Функціональні вимоги для WEB-додатку з продажу та обміну токенізованими активами наведені у таблиці нижче.

Таблиця 1.2 – Функціональні вимоги

Актор	Варіант використання	Функціональна вимога	Пріоритет
Клієнт	Відображення доступних для купівлі книжок	На окремій сторінці мають бути відображені книги, доступні для покупки а також детальна інформація про них.	Високий
Клієнт	Можливість здійснення покупки	Біля кожної доступної для покупки книги має знаходитись кнопка “BUY”, після натискання якої користувач підписує транзакцію та отримує книгу.	Високий
Клієнт	Особистий кабінет	На окремій сторінці мають бути відображені усі наявні книжки користувача, детальна інформація про них та кнопка для завантаження файлу.	Високий
Клієнт	Відображення доступних книжок для обміну	На окремій сторінці мають бути відображені усі наявні пропозиції щодо обміну книжками з усією необхідною для проведення операції інформацією.	Високий

Продовження таблиці 1.2

Клієнт	Можливість здійснення обміну	Біля кожної доступної пропозиції обміну має знаходитись кнопка “EXCHANGE”, після натискання якої користувач підписує необхідні для обміну транзакції та отримує обрану книгу віддаючи певну власну.	Високий
Клієнт	Можливість створення власного запиту на обмін	Користувач повинен мати можливість створити власний запит на обмін віддавши книгу з власної бібліотеки щоб отримати іншу. Підписавши усі необхідні для обміну транзакції запит має буди доданим до загальнодоступного списку.	Високий
Адміністратор	Можливість додавання нових книжок	Користувач, наділений правами адміністратора, повинен мати можливість додати нову книгу. Підписавши усі необхідні для створення транзакції нова книша має буди додана до загальнодоступного каталогу для купівлі.	Високий

Продовження таблиці 1.2

Адміністратор	Можливість редагування інформації про існуючі книги	Користувач, наділений правами адміністратора, повинен мати можливість редагувати ціну та доступну для продажу кількість активних для продажу книг.	Високий

Схема трасування вимог зображена на рисунку 1.3.

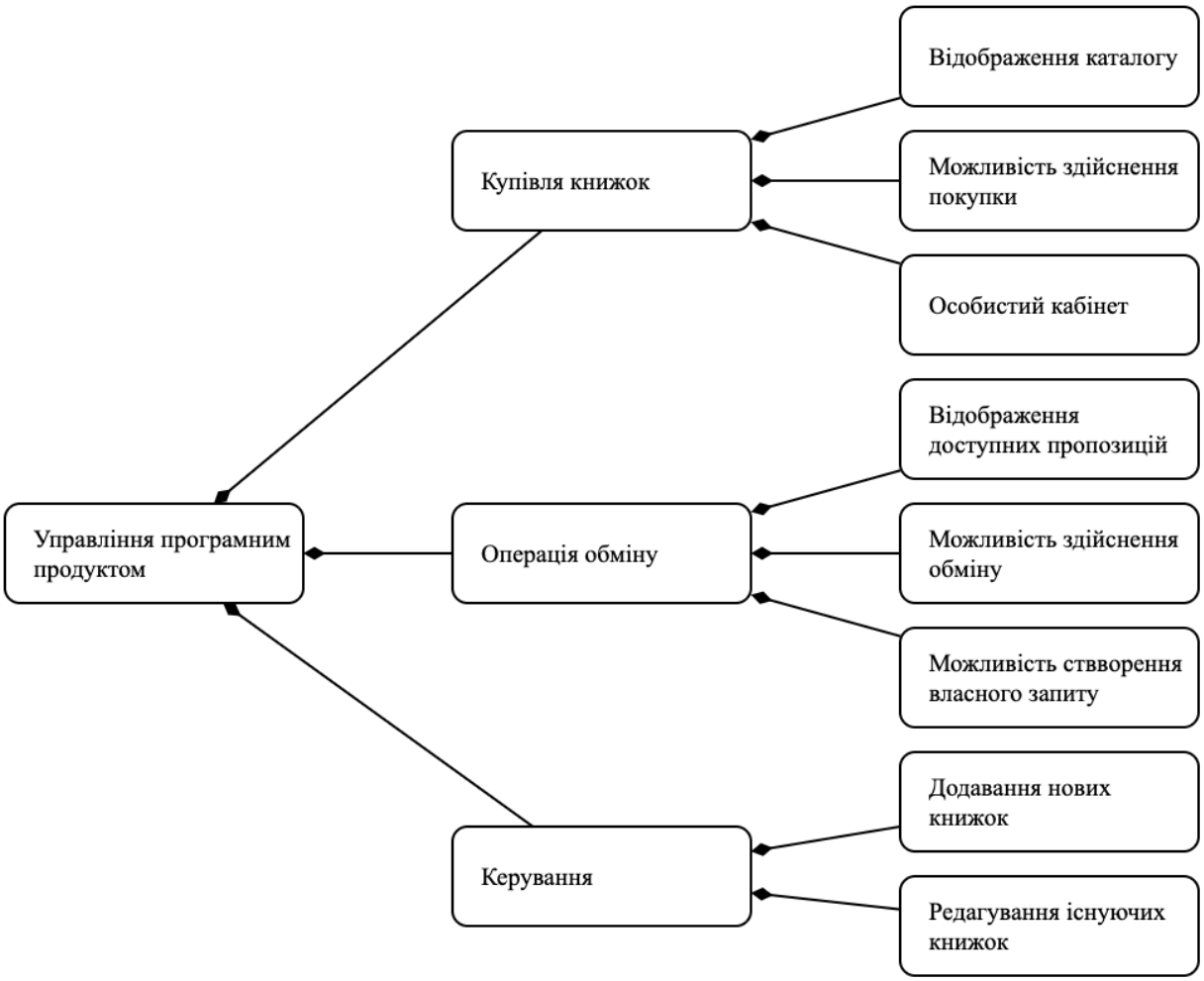


Рисунок 1.3 – Схема трасування вимог

1.4.2 Розроблення нефункціональних вимог

Для користування WEB-додатком необхідно мати доступ до персонального комп’ютера з встановленою операційною системою та веб-

браузером. Персональний комп'ютер повинен мати інтернет підключення з пропускною можливістю не менше 1 mb/s. Користувач повинен мати доступ до власного гаманця у мережі Ethereum та встановлене розширення для браузера MetaMask.

1.4.3 Постановка комплексу завдань модулю

Мета – створити WEB-додаток для купівлі та обміну токенізованими активами книжок. Передбачити можливість адміністрування існуючих активів.

Призначення – купівля та обмін книжками

Задачі – додавання нових книжок, редагування існуючих, перегляд доступного каталогу, здійснення покупки, відображення власних книжок у особистому кабінеті, можливість здійснення обміну та створення власного запиту на обмін книжками.

Розпишемо детальніше кожну з задач.

Додавання нових книжок – задача передбачає створення нового токена , додавання інформації про зміст, автора, жанри, зазначення ціни та кількості доступних екземплярів. До кожної книжки має бути доданий файл розширення .pdf.

Редагування існуючих книжок – адміністратор має можливість змінювати кількість доступних книжок та ціну. Зміни мають бути проведені у смарт контракті відповідною транзакцією. Сервер має автоматично оновити стан бази даних відповідно до нових значень.

Каталог – сторінка має містити інформацію про доступні для купівлі книжки. Мають бути відображені дані як про саму книжку, так і про ціну та доступну для купівлі кількість.

Здійснення покупки – задача передбачає можливість користувачем шляхом підписання відповідної транзакції отримати токен обраної книги. Цей токен дозволить отримати доступ до самої книги та може бути використаний на розсуд покупця.

Особистий кабінет (бібліотека) – сторінка має дані про усі книги, якими володіє даний користувач. Також користувач повинен мати можливість завантаження файлу книги. Для цього йому необхідно верифікувати запит шляхом підписання відповідної транзакції.

Сторінка обміну – має містити усі доступні запити користувачів на обмін. Необхідно відображати достатню інформацію стосовно того, які книги обмінюються.

Здійснення обміну – задача передбачає можливість користувачем шляхом підписання відповідних транзакцій здійснити обмін власної книги на пропоновану. Після проведення операції користувач повністю втрачає доступ до попередньої книги віддавши токен, та отримує повний доступ до нової отримавши відповідний токен.

Створення власного запиту на обмін – користувач повинен мати можливість створити власну пропозицію обміну. На час дії пропозиції користувач втрачає доступ до книги, яку хоче обміняти. Передбачити можливість скасування пропозиції за ініціативи користувача, який її створив.

1.5 Висновки по розділу

В першому розділі було розглянуто існуючі стандарти токенів, обрано той який будемо застосовувати для токенізації, проаналізовано відомі ІТ проєкти пов’язані з даною предметною областю та проведено аналіз вимог до програмного забезпечення. Описані функціональні та не функціональні вимоги.

Під час розгляду аналогів та відомих технічних рішень, акцент був зроблений саме на прикладах токенізації. Було розглянуто як інструменти безпосередньо впровадження технології так і на існуючі продукти. В якості проєктів-прикладів було обрано продукт з фінансової сфери (крипто-гривня) та сфери державних реєстрів (земельний кадастр України). Метою такого вибору стала оцінка того, як одна технологія може бути впроваджена в різних видах діяльності.

Опрацьовуючи аналоги було зроблено висновки про переваги та недоліки впровадження технології та запропоновано власний підхід до реалізації.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для створення якісного програмного продукту необхідно змоделювати бізнес процеси та архітектуру програмного забезпечення. Далі представлені діаграми бізнес процесів:

– схема бізнес-процесу для покупки активу (рисунок 2.1);

– схема бізнес-процесу для обміну активу (рисунок 2.2).

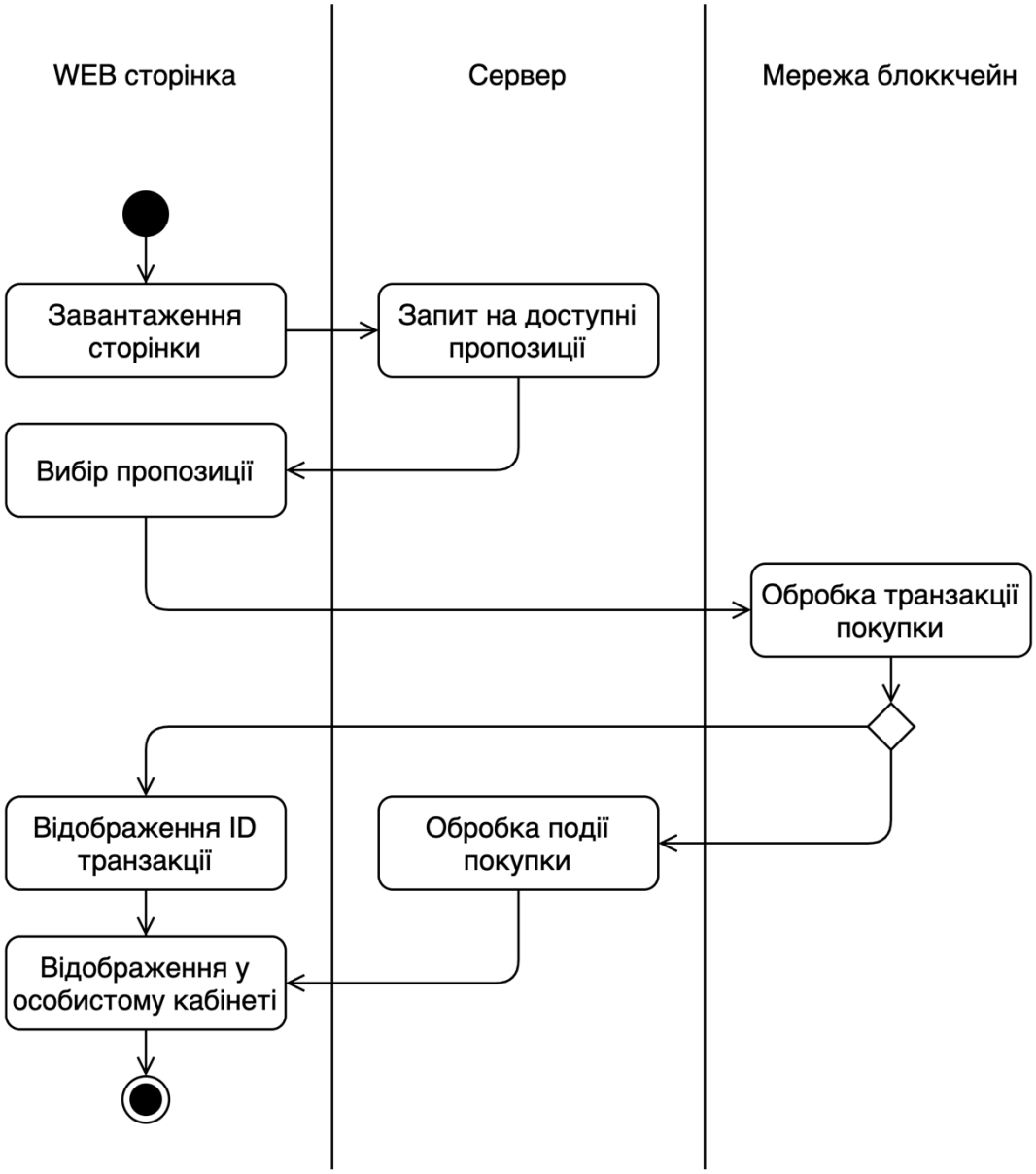


Рисунок 2.1 – Схема бізнес-процесу покупки

Рисунок 2.2 – Схема бізнес-процесу обміну

2.2 Архітектура програмного забезпечення

Для розробки програмного продукту було обрано інтерпретовану мову програмування NodeJS. Оскільки однією з цілей розробки є створення WEB-додатку, то доцільно буде обрати фреймворк express та розширенням ejs. Для балансування навантаження на сервер було застосовано бібліотеку pm2. В якості мови програмування було обрано solidity.

Express передбачає реалізацію на основі архітектури MVT(Model-View-Template), яка є модифікацією відомої у WEB розробці – MVC. Схематично, архітектура MVT представлена на рисунку 2.3.

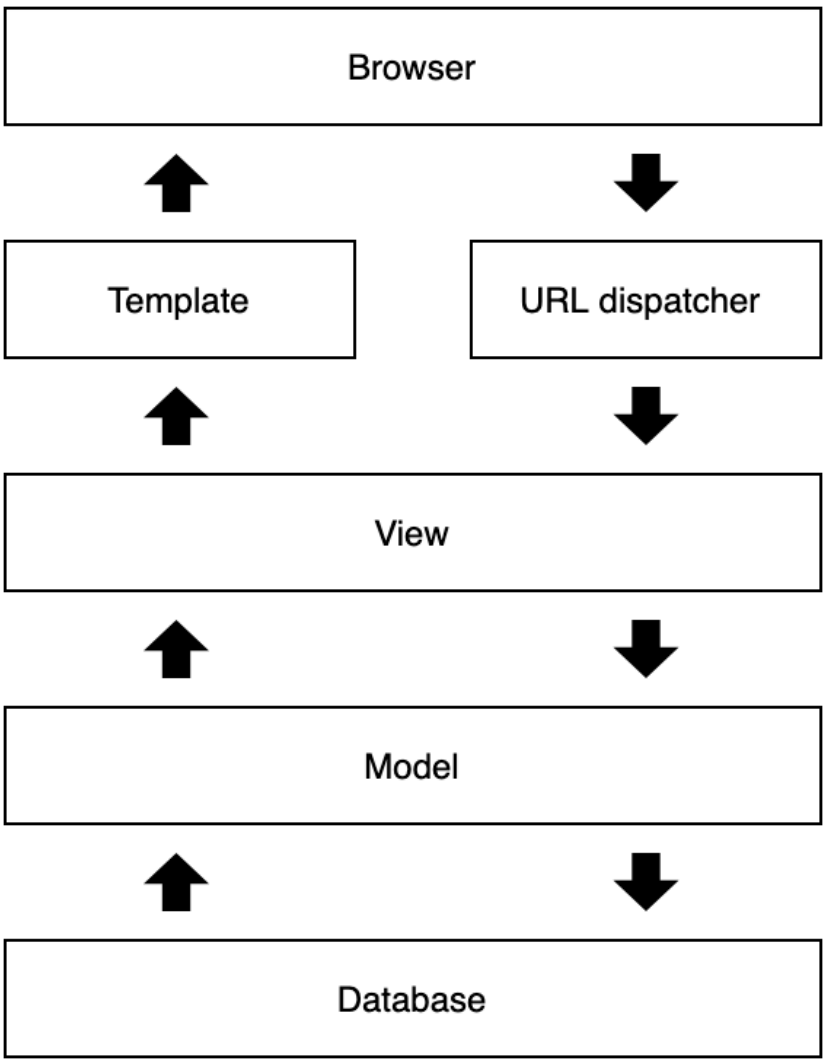


Рисунок 2.3 – Архітектура MVT

Основні елементи:

– URL dispatcher: при отриманні запиту визначає ресурс-обробник аналізуючи URL адресу запиту;

– View: слугує обробником запитів. За необхідності взаємодії з базою даних запит передається через Model. Повертає результат у вигляді шаблону застосовуючи компонент Template;

– Model: описує моделі даних якими працює програмний продукт. Окремі моделі, як правило, відповідають таблицями в базі даних;

– Template: реалізує логіку уявлення у вигляді html розмітки.

Для реалізації інтерфейсу WEB-додатка було застосовано шаблони Bootstrap 4 та бібліотеку jquery. Для взаємодії з блокчейн мережею використовується бібліотека web3. Дана бібліотека створює підключення використовуючи вбудований або вказаний провайдер. Для серверної частини вона додається у вигляді пакету, для частини WEB-додатка – є частиною розширення MetaMask.

У якості провайдера для цієї бібліотеки на стороні сервера використано провайдер Infura, WEB-частина використовує провайдер, вбудований у розширення MetaMask. Даний провайдер є публічним та доступним для використання без додатковий дозволів. В рамках даного продукту обраний провайдер повністю покриває запит до трафіку.

Web3 містить функції для створення, підписання приватним ключем та відправлена підписаної транзакції у мережу. Також у неї входять функції для отримання даних зі сховищ існуючих смарт контрактів. Для перетворень та обробки отриманих даних у бібліотеку входять відповідні утиліти. Дана бібліотека створена розробниками Ethereum та э рекомендованною до використання.

Усі частини програмного продукту працюють з основними сутностями. Виділимо дві – книги та операції обміну. Далі опишемо з чого складаються моделі цих сутностей на рівні смарт-контрактів в таблицях 2.1 та 2.2.

Таблиця 2.1 – Модель книги

Назва поля	Опис	Тип	Обов'язковість
price	Визначає ціну книги	uint256	так
count	Визначає загальну доступну для продажу кількість	uint	так
tokenName	Ідентифікатор серії токена	string	так
status	Статус активності книги	bool	так

Таблиця 2.2 – Модель операції обміну

Назва поля	Опис	Тип	Обов'язковість
orderId	Ідентифікатор операції	uint	так
owner	Визначає власника (того, хто створив) операції	address payable	так
sellTokenId	Ідентифікатор токена для віддавання	uint	так
buyTokenName	Ідентифікатор серії токена для отримання	string	так
status	Статус активності операції	bool	так

Тепер розглянемо ці ж самі моделі з точки зору сервера та бази даних. Моделі описані в таблицях 2.3 та 2.4.

Таблиця 2.3 – Модель книги

Назва поля	Опис	Тип	Обов’язковість
title	Назва книги	varchar(50)	так
image	Посилання на зображення	varchar(200)	ні
short_description	Короткий опис книги	varchar(200)	так
description	Повний опис книги	text	так
year	Рік видання	integer	ні
author	Ім’я та прізвище автора	varchar(100)	так
country	Країна автора	varchar(75)	так
genres	Список жанрів	integer[]	так
price	Ціна книги	bigint	так
count	Загальна доступна для продажу кількість	integer	так
key	Ідентифікатор серії токена	varchar(25)	так
status	Статус активності книги	bool	так

Таблиця 2.4 – Модель операції обміну

Назва поля	Опис	Тип	Обов’язковість
order_id	Ідентифікатор у смарт контракті	integer	так
sell_token_id	Ідентифікатор токена для віддавання	integer	так
buy_book_id	Ідентифікатор книги для отримання	integer	так
status	Статус активності операції	bool	так

На рисунку 2.4 представлена структурна схема розгортання програмного продукту.

Рисунок 2.4 – Схема структурна розгортання

Архітектурно програмний продукт складається з 4 компонентів. Обмін даними між клієнт серверною частиною та блокчейном Ethereum здійснюється через Blockchain provider. У даній реалізації використовуємо публічний сервіс Infura [6], який надає HTTP та WS інтерфейси. Отримані з блокчейн мережі дані сервер зберігає у базі даних через HTTP підключення.

2.3 Конструювання програмного забезпечення

Розглянемо основні функції для виконання бізнес логіки на рівні смарт контрактів. Інформація про назву, призначення та параметри функцій смарт контракту токена представлена у таблиці 2.5.

Таблиця 2.5 – Специфікації функцій смарт контракту токена

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
addMinter	Адреса гаманця		Додавання прав створення нових токенів.
approve	Адреса гаманця та ID токена		Надавання гаманцю прав на користування токеном.
mint	Гаманець отримувача, ID та ім’я нового токена		Створення токена з вказаними параметрами на гаманець отримувача.
mintNext	Гаманець отримувача та ім’я нового токена		Створення токена з вказаними параметрами на гаманець отримувача. ID задається автоматично.
renounceMinter			Зупинення процесу створення нових токенів назавжди.
safeMint	Гаманець отримувача, ID та ім’я нового токена		Більш безпечна версія mint. Проводяться додаткові перевірки параметрів.

Продовження таблиці 2.5

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
transferFrom	Гаманець відправника, отримувача та ID токена		Операція надсилання токену з одного гаманця на інший.
safeTransferFrom	Гаманець відправника, отримувача, ID токена та додаткові параметри		Більш безпечна версія transferFrom. Проводяться додаткові перевірки параметрів.
setApprovalForAll	Гаманець отримувача статус		Надання чи позбавлення права певному гаманцю використовувати усі токени відправника.
balanceOf	Гаманець	Поточний баланс	Отримання загальної кількості токенів, що належить певному гаманцю.
getApproved	ID токена	Гаманець	Отримання інформації про доступ на користування для певного токена.
getApprovedForAll	Гаманець власника та цільовий гаманець	Статус	Отримання інформації про доступ на користування усіма токенами.

Продовження таблиці 2.5

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
isMinter	Цільовий гаманець	Статус	Отримання інформації про доступ до випуску токенів певним гаманцем.
name		Назва контракту	Отримання назви смарт контракту.
ownerOf	ID токена	Адреса власника	Отримання гаманця-власника вказаного токена.
supportInterface	Дані інтерфейсу	Статус	Отримання статусу можливості інтеграції з іншим контрактом за вказаним інтерфейсом.
symbol		Скорочення контракту	Отримання скороченої назви смарт контракту.
tokenByIndex	Індекс токена	ID токена	Отримання ID токена за його індексом.
tokenName	ID токена	Назва токена	Отримання назви токена за його ID.
tokenOfOwnerByIndex	Індекс токена та гаманець власника	ID токена	Отримання ID токена за його індексом та власником.
tokrnURI	ID токена	URI	Отримання URI токена за його ID.

Продовження таблиці 2.5

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
totalSupply		Кількість токенів	Отримання загальної кількості випущених токенів.

Інформація про назву, призначення та параметри функцій смарт контракту магазину представлена у таблиці 2.6.

Таблиця 2.6 – Специфікації функцій смарт контракту магазину

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
transferOwnership	Гаманець нового власника		Передача прав власника контракту.
acceptOwnership			Прийом прав власника контракту.
addItem	Параметри нової книги	ID нової книги	Додавання нових книг
setItemCount	ID книги та нова кількість для продажу		Зміна загальної доступної кількості книг для продажу.
setItemPrice	ID книги та нова ціна		Зміна ціни книги.
buyItem	ID книги		Операція покупки книги.

Продовження таблиці 2.6

setPool	Новий гаманець		Встановлення гаманця для отримання коштів з продажу книг.
setTokensContract	Адреса контракту токенів		Встановлення зв’язку між контрактом токенів та магазину.
setMainStatus	Нове значення статусу		Слугує для аварійного відключення усіх функцій у випадку виявлення критичної помилки.

Інформація про назву, призначення та параметри функцій смарт контракту обміну представлена у таблиці 2.7.

Таблиця 2.7 – Специфікації функцій смарт контракту обміну

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
transferOwnership	Гаманець нового власника		Передача прав власника контракту.
acceptOwnership			Прийом прав власника контракту.
createOrder	Параметри нового запиту на обмін	ID нового запиту	Створення нових запитів на обмін.

Продовження таблиці 2.7

cancelOrder	ID запиту		Скасування запиту на обмін
fillOrder	ID запиту, ID токена на обмін		Проведення операції обміну
buyItem	ID книги		Операція покупки книги.
setTokensContract	Адреса контракту токенів		Встановлення зв’язку між контрактом токенів та обміну.
setMainStatus	Нове значення статусу		Слугує для аварійного відключення усіх функцій у випадку виявлення критичної помилки.

Інформація про назву, призначення та параметри функцій допоміжного смарт контракту для завантаження файлів представлена у таблиці 2.8.

Таблиця 2.8 – Специфікації функцій допоміжного смарт контракту для завантаження файлів

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
requestAccess	ID запиту		Створення запиту на завантаження файлу.

2.4 Аналіз безпеки даних

Безпека даних є невід'ємною частиною будь якої розробки. Необхідно та важливо на кожному етапі забезпечити цілісність та захист даних від несанкціонованих змін.

Основним гарантом безпеки даних виступає смарт контракт. Для внесення будь-яких змін у системі необхідно виконати відповідну функцію у смарт контракті. Оскільки на рівні цих функцій строго перевіряється виконувач операції, то не маючи відповідного приватного ключа неможливо виконати зміни. Таким чином користувач має доступ лише до зміни власних даних.

Сервер та база даних у створеній архітектурі виступають дублюючим сховищем даних зі смарт контрактів. Після виконання тієї чи іншої операції у смарт контракті створюється сигнал про подію зміни. Сервер відслідковує ці сигнали та оновлює стан бази даних. Усі API інтерфейси, які доступні користувачу, працюють лише на читання.

Наявність сервера забезпечую більшу швидкість отримання даних та інструменти для їх системного аналізу. Використання безпосередньо сховища смарт контрактів для цих цілей нераціональне через швидкість доступу. Також з метою економії на комісіях за транзакції дані на рівні смарт контактів зберігаються у спрощеному вигляді. Цього достатньо щоб переконатися у правильності роботи, але такий підхід викликає труднощі з обробкою цих даних. Саме цю проблему вирішує серверна частина.

Таким чином безпека роботи продукту залежить від безпеки самої мережі блокчейн (було обрано відносно надійну) та безпеки створених смарт контрактів. Контракти пройшли тестування, тому їх можна вважати надійними.

2.5 Висновки по розділу

При написанні другого розділу було описано архітектуру програмного продукту, створено схеми бізнес процесів та детально розібрано усі необхідні для роботи додатка функції. Було проаналізовано безпеку розробленої архітектури.

Основними рисами розробленої архітектури стали децентралізація та безпека даних на рівні блокчейн мережі. Обраний підхід до впровадження технології дозволив досягти надійності роботи додатка та захистити дані. Основна ідея полягає у тому, що єдиним джерелом достовірних даних є смарт контракт. Усі можливі зміни чітко визначені створеними функціями та потребують підпису відповідних транзакцій. Таким чином безпека роботи додатка стає рівною безпеці обраної мережі, яку вважаємо надійною.

Іншою характеристикою продукту стала децентралізація. Смарт контракт є загальнодоступним і людина може довести володіння тим чи іншим активом без використання конкретного клієнт-серверного додатка.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Для характеристики якості програмного забезпечення обрано наступні характеристики та під-характеристики якості:

– функціональність:
  а) придатність;
  б) захищеність.
– надійність:
  а) відмово стійкість;
  б) завершеність.
– зручність використання:
  а) зрозумілість;
  б) освоюваність.
– супроводжуваність:
  а) гнучкість;
  б) можливість тестування.
– знаходження попарно конфліктуючих між собою властивостей.

Результати тестування

40 – кількість функцій

2 – кількість відмов

40 – к-сть реалізованих функцій

24 – к-сть використовуваних користувачем функцій

94 – к-сть звернень користувача до системи і даних

104 – к-сть звернень, записаних в лог

2 – к-сть помилок

0 – к-сть вильотів

39 – к-сть пунктів відповідності надійності

12 – к-сть реалізованих пунктів відповідності надійності

36 – к-сть зрозумілих функцій

16 – к-сть змін

82 – Sum (A / B), A – час потрачений на зміну, B – розмір зміни

10 – к-сть тестів

30 – к-сть тестів з вбудованими функціями

Інтегральна шкала для моделі зовнішньої якості

Відмінно верхня межа = 1×0.9 + 1×1 + 1×0.75 + 1×0.65 + 1×0.65 + 1×0.25 + 1×0.7 + 1×0.45 = 6

Відмінно нижня межа = 0.7×0.6 + 0.9×1 + 0.7×0.75 + 1×0.65 + 1×0.65 + 1×0.25 + 0.9×0.6 + 0.7×0.45 = 5.15

Добре верхня межа = 0.7×0.6 + 1×1 + 1×075 + 1×0.65 + 1×0.5 + 1×0.25 + 1×0.65 + 1×0.45 = 5.15

Добре нижня межа = 0.7×0.6 + 0.9×1 + 0.7×0.75 + 1×0.65 + 1×0.5 + 1×0.25 + 0.9×0.6 + 0.7×0.4 = 3.8

Задовільно верхня межа = 1*0.7 + 1*1 + 1*0.8 + 1*0.7 + 1*0.5 + 1*0.3 + 1*0.6 + 1*0.4 = 3.8

Задовільно нижня межа = 0.7×0.75 + 0.9×1 + 0.7×0.75 + 1×0.65 + 1×0.65 + 1×0.25 + 0.9×0.6 + 0.7×0.45 = 2.9

Незадовільно верхня межа = 0.4×0.75 + 1×1 + 1×0.75 + 1×0.65 + 1×0.65 + 1×0.25 + 1×0.6 + 1×0.45 = 2.9

Незадовільно нижня межа = 0×0.75 + 0×1 + 0×0.75 + 0×0.65 = 0

Таблиця 3.7 – Оцінка зовнішньої якості

Відмінно	5.15 – 6
Добре	3.8 – 5.15
Задовільно	2.9 – 3.8
Незадовільно	0 – 2.9

Інтегральна шкала для моделі експлуатаційної якості

Відмінно верхня межа = 1×0.7 + 1×1 + 1×0.85 + 1×0.75 + 1×0.45 = 4.2

Відмінно нижня межа = 0.7×0.7 + 0.9×1 + 0.7×0.85 + 1×0.75 = 3.1

Добре верхня межа = 1×0.7 + 1×1 + 1×0.85 + 1×0.75 + 1×0.55 + 1×0.25 = 3.1

Добре нижня межа = 0.7×0.7 + 0.9×1 + 0.7×0.85 + 1×0.75 + 1×0.55 = 2.2

Задовільно верхня межа = 1×0.7 + 1×1 + 1×0.85 + 1×0.75 + 1×0.55 = 2.2

Задовільно нижня межа = 0,7×0.7 + 0,9×1 + 0,7×0.85 + 1×0.75 = 1.4

Незадовільно верхня межа = 1×0.7 + 1×1 + 1×0.85 + 1×0.75 + 1×0.5 = 1.4

Незадовільно нижня межа = 0×0.7 + 0×1 + 0×0.85 + 0×0.75 = 0

Таблиця 3.8 – Оцінка зовнішньої якості

Відмінно	4.2
Добре	3.1 – 4.2
Задовільно	2.2 – 3.1
Незадовільно	0 – 1.4

Отриманий результат

P = 5.62 + 3.12 = 8.74 – добре

Рівень якості системи: 8.74 / 10.2 = 0,856.

Якість систем складає 86% від запланованої.

3.2 Опис процесів тестування

Під час тестування в першу чергу слід звернути увагу на наступне:

– належний рівень безпеки даних;

– відповідність дизайну до вимог технічного дизайну;

– правильна робота елементів інтерфейсу;

– інтуїтивність користувацького інтерфейсу;

– коректність результатів роботи додатка.

Для тестування обраний метод Gray Box Testing. Перевіряється код та безпосередньо сам додаток на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

– тестування продуктивності;

– функціональне тестування;

– тестування стабільності;

– тестування інтерфейсу;

Працездатність перевіряється наступним чином:

– динамічне ручне тестування – введення коректних та некоректних даних;

– динамічного ручного тестування на відповідність продукту функціональним вимогам;

– статичне тестування коду;

– тестування елементів інтерфейсу.

3.3 Опис контрольного прикладу

Під час тестування було повністю перевірено функціональність додатка. Послідовно були розібрані та протестовані варіанти використання та проаналізовані отримані результати. Детальний опис у наступних таблицях:

– здійснення покупки книги (таблиця 3.9);

– перегляд власних книжок (таблиця 3.10);

– створення запиту на обмін (таблиця 3.11);

– виконання запиту на обмін (таблиця 3.12).

Таблиця 3.9 – Перевірка операції покупки книги

Мета тесту	Перевірка операції покупки книги
Початковий стан	Відкрита сторінка каталогу та виконаний вхід через MetaMask.
Вхідні дані	
Схема проведення тесту	Обрати книгу з каталогу. Натиснути кнопку покупки та підписати відповідну транзакцію.
Очікуваний результат	Токен обраної книги перейде у власність гаманця користувача. На сторінці власної бібліотеки можна буде побачити куплену книгу.
Стан програмного продукту після проведення випробувань	У випадку успішної покупки книга додається до гаманця користувача. Кількість доступних пропозицій зменшується на одну.

Таблиця 3.10 – Перегляд власних книжок

Мета тесту	Перевірка правильності відображення сторінки бібліотеки користувача.
Початковий стан	Відкрита сторінка бібліотеки та виконаний вхід через MetaMask.
Вхідні дані	
Схема проведення тесту	Перевірити правильність відображення бібліотеки виходячи із того, які токени належать гаманцю користувача.
Очікуваний результат	На сторінці відображаються усі книги, якими володіє користувач.
Стан програмного продукту після проведення випробувань	На сторінці відображаються усі книги, якими володіє користувач.

Таблиця 3.11 – Перевірка операції створення запиту на обмін

Мета тесту	Перевірка правильності роботи операції створення запиту на обмін книгами.
Початковий стан	Відкрита сторінка обмінника та виконаний вхід через MetaMask.
Вхідні дані	Книга на обмін
Схема проведення тесту	Користуючись інтерфейсом користувач додає власну книгу, та обирає бажану. Даль відбувається підтвердження та підписання відповідної транзакції.
Очікуваний результат	Запит буде доданий до смарт контрактів та почне відображатися для усіх користувачів. Книга буде недоступна у бібліотеці.

Продовження таблиці 3.11

Стан програмного продукту після проведення випробувань	У випадку успіху операції стан токена тієї книги, які віддає користувач зміниться і перейде у тимчасове розпорядження смарт контрактів. Операція обміну буде додана до смарт контрактів та стане загальнодоступною.

Таблиця 3.12 – Перевірка операції виконання запиту на обмін

Мета тесту	Перевірка правильності роботи операції обміну книгами.
Початковий стан	Відкрита сторінка обмінника та виконаний вхід через MetaMask.
Вхідні дані	Пропозиція обміну
Схема проведення тесту	Користувач обирає доступну операцію обміну. Далі відбувається підписання відповідних транзакції для її завершення.
Очікуваний результат	Обмін книгами відбувся. У обох користувачів у бібліотеці з’явилась нова книга і повністю зникла стара.
Стан програмного продукту після проведення випробувань	Токени двох книг, які беруть участь у обміні змінюють власників. Операція обміну переходить у стан виконаної.

3.4 Висновки по розділу

При написанні третього розділу протестована створена архітектура програмного продукту, визначені критерії надійності та розрахований коефіцієнт. Вважаємо отриманий результат (86%) достатнім для даного типу продукту.

Основні причини проблем полягають у принципах роботи самої мережі. Відносна повільність роботи блокчейн підвищує ймовірність помилок. Важливо відмітити, що помилки не призводять до втрати чи несанкціонованих змін даних, а є причиною відхилення операції. При повторному запиті операція виконується.

Також було проведено тести основних механік додатка. Результати збігаються з очікуваними та підтверджують правильність роботи програмного продукту.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розгортання програмного продукту складається з двох частин. У першій відбувається розгортання смарт контрактів. У другий встановлюється клієнт-серверна частина.

Почнемо з розгортання смарт контрактів.

– Смарт контракт токена:

а) завантажуємо у середовище для роботи з контрактами наступні файли:

1) Address.sol;
2) Context.sol;
3) Counters.sol;
4) ERC165.sol;
5) ERC721.sol;
6) ERC721Enumerable.sol;
7) ERC721Full.sol;
8) ERC721Metadata.sol;
9) ERC721Mintable.sol;
10) IERC721Receiver.sol;
11) MinterRole.sol;
12) Roles.sol;
13) SafeMath.sol.

б) компілюємо файл ERC721Full.sol;

в) проводимо розгортання у мережі з наступними параметрами:

1) name = “Smart books”;
2) symbol = “SB”.

г) отримаємо адресу смарт контракту та запам’ятовуємо її.

– Смарт контракт магазину:
  а) завантажуємо у середовище для роботи з контрактами наступні файли:
    1) Store.sol;
    2) Roles.sol.
  б) компілюємо файл Store.sol.
  в) проводимо розгортання у мережі з наступними параметрами:
    1) contractAddress = < ERC721Full >.
  г) отримаємо адресу смарт контракту та запам’ятовуємо її.
– Смарт контракт обмінника:
  а) завантажуємо у середовище для роботи з контрактами наступні файли:
    1) Exchange.sol;
    2) Roles.sol.
  б) компілюємо файл Exchange.sol;
  в) проводимо розгортання у мережі з наступними параметрами:
    1) contractAddress = < ERC721Full >.
  г) отримаємо адресу смарт контракту та запам’ятовуємо її.
– Смарт контракт файлів:
  а) завантажуємо у середовище для роботи з контрактами наступні файли:
    1) Files.sol.
  б) компілюємо файл Files.sol;
  в) проводимо розгортання у мережі:
  г) отримаємо адресу смарт контракту та запам’ятовуємо її.

Після розгортання смарт контрактів необхідно встановити клієнт-серверний додаток.

– Встановлення необхідних пакетів:
  а) встановлюємо NodeJS та npm;
  б) встановлюємо yarn;
  в) встановлюємо pm2.
– Розгортання бази даних:
  а) встановлюємо PostgreSQL;
  б) створюємо користувача.
– Розгортання проєкту:
  а) клонуємо гіт репозиторій;
  б) налаштовуємо файл process.yml та database.json згідно README;
  в) встановлюємо пакети командою yarn;
  г) проводимо міграцію бази даних командою yarn migrate;
  д) запускаємо сервер командою yarn start;
  е) підключаємось до виводу логів командою yarn logs.

4.2 Робота з програмним забезпеченням

Для роботи з програмним забезпеченням необхідно встановити додаток MetaMask [5]. Цей додаток дозволить підключитись до мережі Ethereum, та створити власний гаманець. Використовуватись даний додаток буде переважно для підпису транзакцій.

Після запуску клієнт-серверної частини додатка переходимо на http://localhost:4010/. Основний каталог додатку можемо бачити на рисунку 4.1.

Рисунок 4.1 – Вигляд основного каталогу

Далі користувач обирає книгу, яку хоче придбати. Для цього треба натиснути “Read more” на обраному блоці з зображенням книги. Після цього відкривається модальне вікно з додатковою інформацією – рисунок 4.2.

Рисунок 4.2 – Додаткова інформація про книгу

Наступним кроком користувач купує книгу. Для цього необхідно натиснути кнопку “Buy” та підписати транзакцію покупки – рисунок 4.3.

Рисунок 4.3 – Операція покупки книги

Після підписання транзакції та її підтвердження зі сторони мережі відповідна книга з’явиться у особистій бібліотеці. Посилання знаходиться у навігаційному барі. Вигляд особистої бібліотеки на рисунку 4.4.

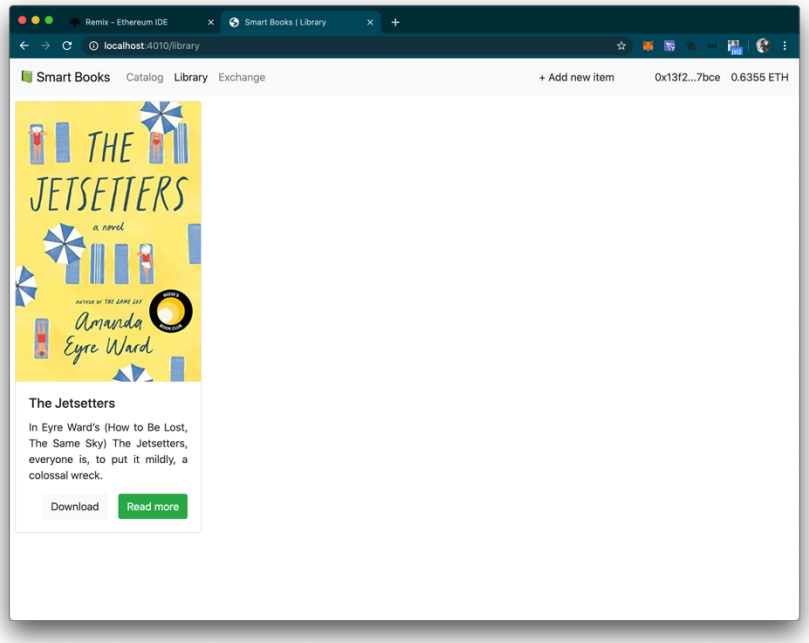


Рисунок 4.4 – Особиста бібліотека

Наступною розглянемо механіку роботи обміну книгами. Переходимо на сторінку “Exchange” (посилання знаходиться у верхньому навігаційному барі). Вигляд сторінки обміну представлений на рисунку 4.5. У верхній частині представвлені власні запити на обмін. У нижній доступні запити від інших користувачів.

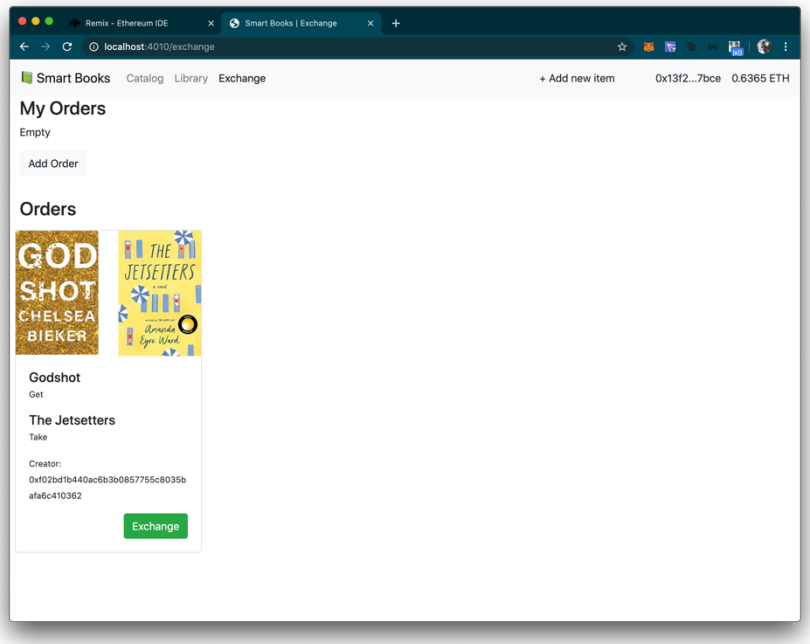


Рисунок 4.5 – Сторінка обміну книгами

Наступним кроком виконаємо запит на обмін. Для цього натискаємо кнопку “Exchange” біля доступної пропозиції. Після цього необхідно підписати дві транзакції – одна для того, щоб смарт контракт обмінника отримав доступ до токена користувача, а інша для створення власне запиту. Дивись рисунок 4.6.

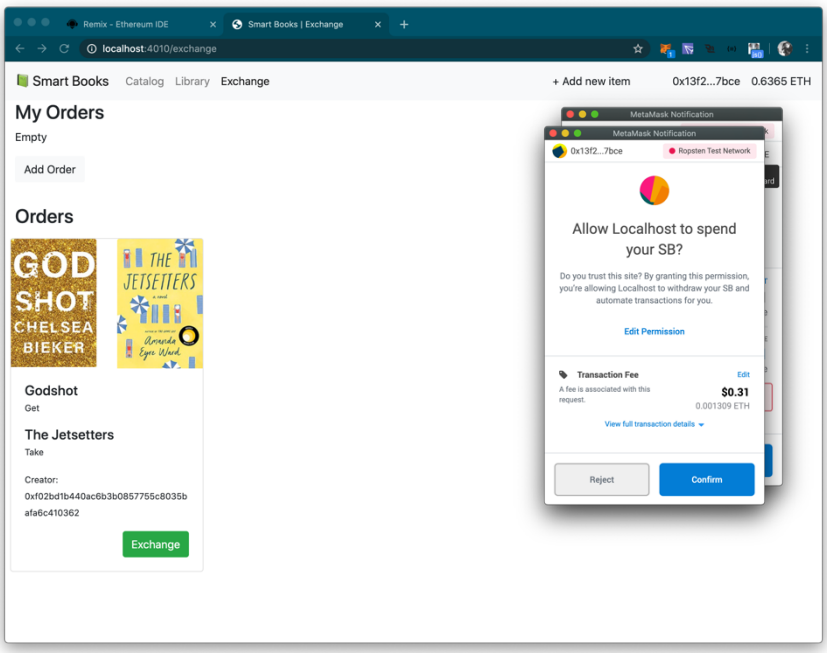


Рисунок 4.6 – Виконання операції обміну

Після підписання транзакцій бажана книга з’явиться у особистій бібліотеці, а віддана – зникне (рисунок 4.7).

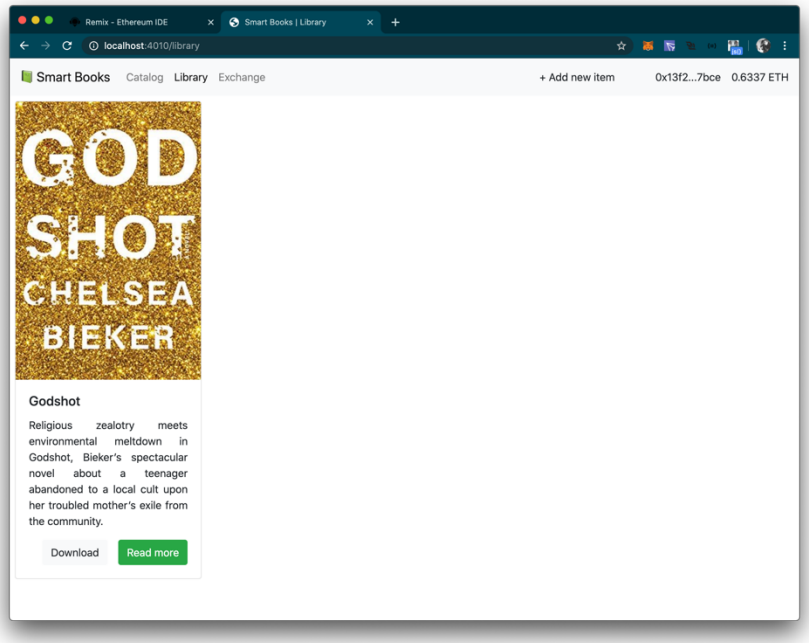


Рисунок 4.7 – Результат обміну

Останнім кроком продемонструємо створення власного запиту. Для цього на сторінці обміну необхідно натиснути “Add Order” та заповнити форму. Далі підписуємо знову дві транзакції – рисунок 4.8.

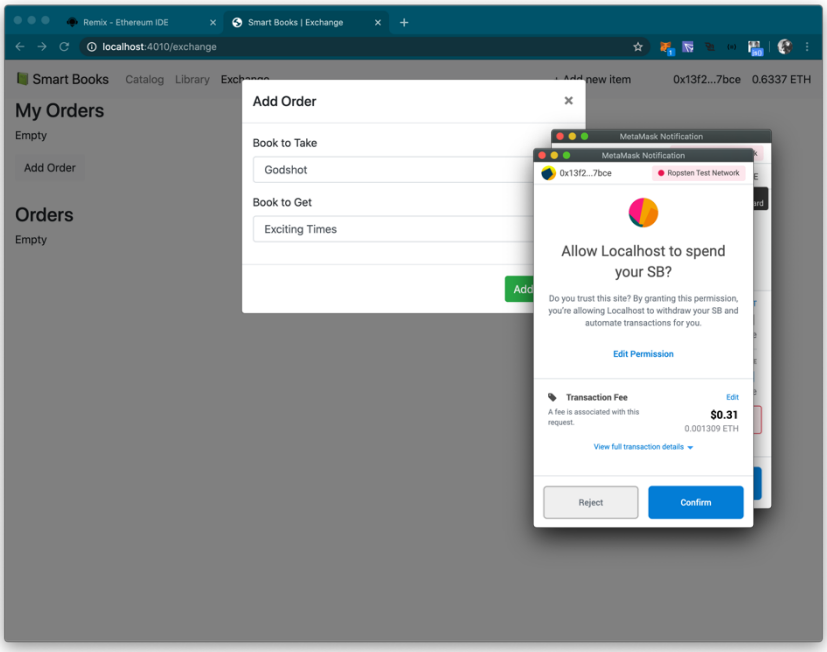


Рисунок 4.8 – Створення запиту на обмін

Після створення запиту він з’явиться у верхній частині сторінки запитів на обмін та буде доступний усім користувачам.

Рисунок 4.9 – Власний запит на обмін

ВИСНОВКИ

У даній дипломній роботі розроблено та реалізовано програмне забезпечення для продажу та обміну токенізованими активами. У якості активів були обрані книги. Програмний продукт можна використовувати як за прямим призначенням (купівля, продаж та обмін книгами), так і в якості прикладу проведення токенізації активів з використанням смарт контрактів на мережі Ethereum.

Програмний продукт складається з трьох частин. Перша – смарт контракти, розгорнуті у мережі Ethereum. Друга – серверна частина для більш швидкого доступу користувачів до даних у смарт контрактах. Третя – інтерфейс користувача.

У першому розділі було розглянуто різні підходи до впровадження токенізації, проаналізовано їх переваги та недоліки. Результатом стало створення вимог до програмного забезпечення та розробка функціональних та нефункціональних вимог.

У другому розділі описано архітектуру програмного продукту, створено схеми бізнес процесів. Також визначено та детально проаналізовано перелік необхідних для роботи додатка функцій для смарт контрактів.

У третьому розділі проаналізовано якість програмного забезпечення та проведено тестування основних механік.

У четвертому розділі описано порядок розгортання застосунку, включаючи смарт контракти, серверну та клієнтську частини.

Результатом даної дипломної роботи є старений застосунок, який можна використовувати як за прямим призначенням, так і в якості демонстрації підходу до впровадження технології блокчейн.

ПЕРЕЛІК ПОСИЛАНЬ

1) Ethereum is a global, open-source platform for decentralized applications: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://ethereum.org;
2) TokenD – a processing center for digital assets: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://tokend.io;
3) Криптогривня: біржа Kuna запустила бета-версію криптовалюти UAX: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://nachasi.com/2020/02/21/cryptohryvnia-uax/;
4) Публічна кадастрова карта України: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://map.land.gov.ua;
5) A crypto wallet & gateway to blockchain apps: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://metamask.io;
6) Ethereum & IPFS APIs. Develop now on Web 3.0: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://infura.io;
7) Building amazing things. Develop now on Web 3.0: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://www.npmjs.com;
8) NodeJS Docs: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://nodejs.org/en/docs/;
9) Safe, stable, reproducible projects: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://yarnpkg.com;
10) Quick start: навчальні матеріали онлайн: [Електрон. ресурс]. – Режим доступу: https://pm2.keymetrics.io/docs/usage/quick-start/.

2тутФакультет інформатики та обчислювальної техніки Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*
(підпис) (ім’я, прізвище)

“___” _____ 2020 р.

Веб-платформа з продажу та обміну токенізованих активів

Опис програми

КПІ.ІП-6113.045440.02.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Д.Ю. Кролевецький

Київ – 2020 року

Тексти програмного коду

Веб-платформа з продажу та обміну токенізованих активів

(Найменування програми (документа))

DVD-R

(Вид носія даних)

12 арк, 27,2 Мб

(Обсяг програми (документа) , арк.,) Мб)

Київ - 2020

Файл Address.sol

```
pragma solidity ^0.5.17;

/**
 * @dev Collection of functions related to the address type.
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the execution of a contract's constructor,
     * its address will be reported as not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this function returns false is an externally-owned
     * account (EOA) and not a contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in construction, since the code is only
        // stored at the end of the constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts and
        // 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned for accounts without code,
        // i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;

        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is simply a type cast: the actual underlying
     * value is not changed.
     *
     * NOTE: This is a feature of the next version of OpenZeppelin Contracts.
     * @dev Get it via `npm install @openzeppelin/contracts@next`.
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
```

```
    }
}
```

Файл Context.sol

```
pragma solidity ^0.5.17;

/*
 * @dev Provides information about the current execution context, including the sender of the transaction and its data.
 * While these are generally available via msg.sender and msg.data, they should not be accessed in such a direct manner,
 * since when dealing with GSN meta-transactions the account sending and paying for execution may not be the actual
 * sender (as far as an application is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying an instance of this contract, which
    // should be used via inheritance.
    constructor () internal { }

    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        // silence state mutability warning without generating bytecode - see
        // https://github.com/ethereum/solidity/issues/2691
        this;
        return msg.data;
    }
}
```

Файл Counters.sol

```
pragma solidity ^0.5.17;

import "./SafeMath.sol";

/**
 * @title Counters
```

```
 * @dev Provides counters that can only be incremented or decremented by one. This can be used e.g. to track the number
 * of elements in a mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;` since it is not possible to overflow a 256 bit integer with
 * increments of one, `increment` can skip the {SafeMath} overflow check, thereby saving gas. This does assume however
 * correct usage, in that the underlying `_value` is never directly accessed.
 */
library Counters {
    using SafeMath for uint256;

    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must be restricted to
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a proposal to add
        // this feature: see https://github.com/ethereum/solidity/issues/4637.
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        // The {SafeMath} overflow check can be skipped here, see the comment at the top.
        counter._value += 1;
    }

    function decrement(Counter storage counter) internal {
        counter._value = counter._value.sub(1);
    }
}
```

Файл ERC165.sol

```
pragma solidity ^0.5.17;

/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts may inherit from this and call {_registerInterface} to declare their support of an interface.
 */
contract ERC165 {
    // bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7.
```

```
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

    // @dev Mapping of interface ids to whether or not it's supported.
    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor () internal {
        // Derived contracts need only register support for their own interfaces, we register support for ERC165 itself.
        _registerInterface(_INTERFACE_ID_ERC165);
    }

    /**
     * @dev See {IERC165-supportsInterface}.
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by `interfaceId`. Support of the actual.
     * ERC165 interface is automatic and registering its interface id is not required.
     *
     * See {IERC165-supportsInterface}.
     *
     * Requirements:
     * - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).
     */
    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
        _supportedInterfaces[interfaceId] = true;
    }
}
```

Файл ERC721.sol

```
pragma solidity ^0.5.17;

import "./Context.sol";
import "./IERC721Receiver.sol";
import "./SafeMath.sol";
import "./Address.sol";
import "./Counters.sol";
import "./ERC165.sol";
```

```
/**
 * @title ERC721 Non-Fungible Token Standard basic implementation
 * @dev see https://eips.ethereum.org/EIPS/eip-721
 */
contract ERC721 is Context, ERC165 {
    using SafeMath for uint256;
    using Address for address;
    using Counters for Counters.Counter;

    event Transfer(address indexed from, address indexed to, uint256 indexed tokenId);
    event Approval(address indexed owner, address indexed approved, uint256 indexed tokenId);
    event ApprovalForAll(address indexed owner, address indexed operator, bool approved);

    // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))` which can be also
obtained as
    // `IERC721Receiver(0).onERC721Received.selector`
    bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;

    // Mapping from token ID to owner
    mapping (uint256 => address) private _tokenOwner;

    // Mapping from token ID to token name
    mapping (uint256 => string) private _tokenName;

    // Mapping from token ID to approved address
    mapping (uint256 => address) private _tokenApprovals;

    // Mapping from owner to number of owned token
    mapping (address => Counters.Counter) private _ownedTokensCount;

    // Mapping from owner to operator approvals
    mapping (address => mapping (address => bool)) private _operatorApprovals;

    /*
     * bytes4(keccak256('balanceOf(address)')) == 0x70a08231
     * bytes4(keccak256('ownerOf(uint256)')) == 0x6352211e
     * bytes4(keccak256('approve(address,uint256)')) == 0x095ea7b3
     * bytes4(keccak256('getApproved(uint256)')) == 0x081812fc
     * bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465
     * bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5
     * bytes4(keccak256('transferFrom(address,address,uint256)')) == 0x23b872dd
     * bytes4(keccak256('safeTransferFrom(address,address,uint256)')) == 0x42842e0e
     * bytes4(keccak256('safeTransferFrom(address,address,uint256,bytes)')) == 0xb88d4fde
     *
     * => 0x70a08231 ^ 0x6352211e ^ 0x095ea7b3 ^ 0x081812fc ^
```

```
     *     0xa22cb465 ^ 0xe985e9c ^ 0x23b872dd ^ 0x42842e0e ^ 0xb88d4fde == 0x80ac58cd
     */
    bytes4 private constant _INTERFACE_ID_ERC721 = 0x80ac58cd;

    constructor () public {
        // register the supported interfaces to conform to ERC721 via ERC165
        _registerInterface(_INTERFACE_ID_ERC721);
    }

    function tokenName(uint256 tokenId) public view returns (string memory) {
        return _tokenName[tokenId];
    }

    /**
     * @dev Gets the balance of the specified address.
     * @param owner address to query the balance of.
     * @return uint256 representing the amount owned by the passed address.
     */
    function balanceOf(address owner) public view returns (uint256) {
        require(owner != address(0), "ERC721: balance query for the zero address");

        return _ownedTokensCount[owner].current();
    }

    /**
     * @dev Gets the owner of the specified token ID.
     * @param tokenId uint256 ID of the token to query the owner of.
     * @return address currently marked as the owner of the given token ID.
     */
    function ownerOf(uint256 tokenId) public view returns (address) {
        address owner = _tokenOwner[tokenId];
        require(owner != address(0), "ERC721: owner query for nonexistent token");

        return owner;
    }

    /**
     * @dev Approves another address to transfer the given token ID. The zero address indicates there is no
approved
     * address. There can only be one approved address per token at a given time. Can only be called by the
token owner
     * or an approved operator.
     * @param to address to be approved for the given token ID.
     * @param tokenId uint256 ID of the token to be approved.
     */
    function approve(address to, uint256 tokenId) public {
```

```
        address owner = ownerOf(tokenId);
        require(to != owner, "ERC721: approval to current owner");

        require(_msgSender() == owner || isApprovedForAll(owner, _msgSender()),
            "ERC721: approve caller is not owner nor approved for all"
        );

        _tokenApprovals[tokenId] = to;
        emit Approval(owner, to, tokenId);
    }

    /**
     * @dev Gets the approved address for a token ID, or zero if no address set. Reverts if the token ID does
not exist.
     * @param tokenId uint256 ID of the token to query the approval of.
     * @return address currently approved for the given token ID.
     */
    function getApproved(uint256 tokenId) public view returns (address) {
        require(_exists(tokenId), "ERC721: approved query for nonexistent token");

        return _tokenApprovals[tokenId];
    }

    /**
     * @dev Sets or unsets the approval of a given operator. An operator is allowed to transfer all tokens of the
sender
     * on their behalf.
     * @param to operator address to set the approval.
     * @param approved representing the status of the approval to be set.
     */
    function setApprovalForAll(address to, bool approved) public {
        require(to != _msgSender(), "ERC721: approve to caller");

        _operatorApprovals[_msgSender()][to] = approved;
        emit ApprovalForAll(_msgSender(), to, approved);
    }

    /**
     * @dev Tells whether an operator is approved by a given owner.
     * @param owner owner address which you want to query the approval of
     * @param operator operator address which you want to query the approval of.
     * @return bool whether the given operator is approved by the given owner.
     */
    function isApprovedForAll(address owner, address operator) public view returns (bool) {
        return _operatorApprovals[owner][operator];
    }
```

```
    /**
     * @dev Transfers the ownership of a given token ID to another address. Usage of this method is discouraged, use
     * {safeTransferFrom} whenever possible. Requires the msg.sender to be the owner, approved, or operator.
     * @param from current owner of the token.
     * @param to address to receive the ownership of the given token ID.
     * @param tokenId uint256 ID of the token to be transferred.
     */
    function transferFrom(address from, address to, uint256 tokenId) public {
        //solhint-disable-next-line max-line-length
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor approved");

        _transferFrom(from, to, tokenId);
    }

    /**
     * @dev Safely transfers the ownership of a given token ID to another address. If the target address is a contract,
     * it must implement {IERC721Receiver-onERC721Received}, which is called upon a safe transfer, and return the magic
     * value `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise, the transfer is
     * reverted. Requires the msg.sender to be the owner, approved, or operator.
     * @param from current owner of the token.
     * @param to address to receive the ownership of the given token ID.
     * @param tokenId uint256 ID of the token to be transferred.
     */
    function safeTransferFrom(address from, address to, uint256 tokenId) public {
        safeTransferFrom(from, to, tokenId, "");
    }

    /**
     * @dev Safely transfers the ownership of a given token ID to another address. If the target address is a contract,
     * it must implement {IERC721Receiver-onERC721Received}, which is called upon a safe transfer, and return the magic
     * value `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise, the transfer is
     * reverted. Requires the _msgSender() to be the owner, approved, or operator.
     * @param from current owner of the token.
     * @param to address to receive the ownership of the given token ID.
     * @param tokenId uint256 ID of the token to be transferred.
     * @param _data bytes data to send along with a safe transfer check.
     */
```

```
    function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data) public {
        require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not owner nor
approved");
        _safeTransferFrom(from, to, tokenId, _data);
    }

    /**
     * @dev Safely transfers the ownership of a given token ID to another address. If the target address is a
contract,
     * it must implement `onERC721Received`, which is called upon a safe transfer, and return the magic
value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise, the transfer is
reverted.
     * Requires the msg.sender to be the owner, approved, or operator.
     * @param from current owner of the token.
     * @param to address to receive the ownership of the given token ID.
     * @param tokenId uint256 ID of the token to be transferred.
     * @param _data bytes data to send along with a safe transfer check.
     */
    function _safeTransferFrom(address from, address to, uint256 tokenId, bytes memory _data) internal {
        _transferFrom(from, to, tokenId);
        require(_checkOnERC721Received(from, to, tokenId, _data), "ERC721: transfer to non
ERC721Receiver implementer");
    }

    /**
     * @dev Returns whether the specified token exists.
     * @param tokenId uint256 ID of the token to query the existence of.
     * @return bool whether the token exists.
     */
    function _exists(uint256 tokenId) internal view returns (bool) {
        address owner = _tokenOwner[tokenId];
        return owner != address(0);
    }

    /**
     * @dev Returns whether the given spender can transfer a given token ID.
     * @param spender address of the spender to query.
     * @param tokenId uint256 ID of the token to be transferred.
     * @return bool whether the msg.sender is approved for the given token ID, is an operator of the owner, or
is the
     * owner of the token.
     */
    function _isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (bool) {
        require(_exists(tokenId), "ERC721: operator query for nonexistent token");
        address owner = ownerOf(tokenId);
```

```
        return (spender == owner || getApproved(tokenId) == spender || isApprovedForAll(owner, spender));
    }

    /**
     * @dev Internal function to safely mint a new token.
     * Reverts if the given token ID already exists. If the target address is a contract, it must implement
     * `onERC721Received`, which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise, the transfer is
reverted.
     * @param to The address that will own the minted token.
     * @param tokenId uint256 ID of the token to be minted.
     */
    function _safeMint(address to, uint256 tokenId, string memory tokenName) internal {
        _safeMint(to, tokenId, tokenName, "");
    }

    /**
     * @dev Internal function to safely mint a new token.
     * Reverts if the given token ID already exists. If the target address is a contract, it must implement
     * `onERC721Received`, which is called upon a safe transfer, and return the magic value
     * `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`; otherwise,  the transfer is
reverted.
     * @param to The address that will own the minted token.
     * @param tokenId uint256 ID of the token to be minted.
     * @param _data bytes data to send along with a safe transfer check.
     */
    function _safeMint(address to, uint256 tokenId, string memory tokenName, bytes memory _data) internal
{
        _mint(to, tokenId, tokenName);
        require(_checkOnERC721Received(address(0), to, tokenId, _data), "ERC721: transfer to non
ERC721Receiver implementer");
    }

    /**
     * @dev Internal function to mint a new token.
     * Reverts if the given token ID already exists.
     * @param to The address that will own the minted token.
     * @param tokenId uint256 ID of the token to be minted.
     */
    function _mint(address to, uint256 tokenId, string memory tokenName) internal {
        require(to != address(0), "ERC721: mint to the zero address");
        require(!_exists(tokenId), "ERC721: token already minted");

        _tokenOwner[tokenId] = to;
        _tokenName[tokenId] = tokenName;
        _ownedTokensCount[to].increment();
```

```
    emit Transfer(address(0), to, tokenId);
}

/**
 * @dev Internal function to burn a specific token.
 * Reverts if the token does not exist. Deprecated, use {_burn} instead.
 * @param owner owner of the token to burn.
 * @param tokenId uint256 ID of the token being burned.
 */
function _burn(address owner, uint256 tokenId) internal {
    require(ownerOf(tokenId) == owner, "ERC721: burn of token that is not own");

    _clearApproval(tokenId);

    _ownedTokensCount[owner].decrement();
    _tokenOwner[tokenId] = address(0);

    emit Transfer(owner, address(0), tokenId);
}

/**
 * @dev Internal function to burn a specific token.
 * Reverts if the token does not exist.
 * @param tokenId uint256 ID of the token being burned.
 */
function _burn(uint256 tokenId) internal {
    _burn(ownerOf(tokenId), tokenId);
}

/**
 * @dev Internal function to transfer ownership of a given token ID to another address.
 * As opposed to {transferFrom}, this imposes no restrictions on msg.sender.
 * @param from current owner of the token.
 * @param to address to receive the ownership of the given token ID.
 * @param tokenId uint256 ID of the token to be transferred.
 */
function _transferFrom(address from, address to, uint256 tokenId) internal {
    require(ownerOf(tokenId) == from, "ERC721: transfer of token that is not own");
    require(to != address(0), "ERC721: transfer to the zero address");

    _clearApproval(tokenId);

    _ownedTokensCount[from].decrement();
    _ownedTokensCount[to].increment();
```

```
        _tokenOwner[tokenId] = to;

        emit Transfer(from, to, tokenId);
    }

    /**
     * @dev Internal function to invoke {IERC721Receiver-onERC721Received} on a target address.
     * The call is not executed if the target address is not a contract. This function is deprecated.
     * @param from address representing the previous owner of the given token ID.
     * @param to target address that will receive the tokens.
     * @param tokenId uint256 ID of the token to be transferred.
     * @param _data bytes optional data to send along with the call.
     * @return bool whether the call correctly returned the expected magic value.
     */
    function _checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory _data)
        internal returns (bool)
    {
        if (!to.isContract()) {
            return true;
        }

        bytes4 retval = IERC721Receiver(to).onERC721Received(_msgSender(), from, tokenId, _data);
        return (retval == _ERC721_RECEIVED);
    }

    /**
     * @dev Private function to clear current approval of a given token ID.
     * @param tokenId uint256 ID of the token to be transferred.
     */
    function _clearApproval(uint256 tokenId) private {
        if (_tokenApprovals[tokenId] != address(0)) {
            _tokenApprovals[tokenId] = address(0);
        }
    }
}


});
```

Факультет інформатики та обчислювальної техніки Кафедра автоматизованих систем обробки інформації і управління

"ЗАТВЕРДЖЕНО"

В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*

(підпис) (ім'я, прізвище)

"___" _____ 2020 р.

Веб-платформа для продажу та обміну токенізованих активів

Технічне завдання

КПІ.ІП-6113.045440-03-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Д.Ю. Кролевецький

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-платформа для продажу та обміну токенізованих активів.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного забезпечення «Smart Books» [КПІ.ІП-6113.045440-03-91], котра використовується для продажу та обміну токенізованими активами та призначена функціонування платформи з продажу токенізованих книжок та може слугувати демонстрацією того, як схожі типи активів можна токенізувати.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Веб-платформи «Smart Books» є завдання на дипломне проєктування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для надання можливості користувачу користуватися токенізованими активами та для демонстрації переваг такого підходу.

Метою розробки є створення веб-платформи для продажу та обміну токенізованими активами книжок.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вимоги до функціональних характеристик

4.1.1. Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1. Для користувача:

– покупка активів;
– отримання доступу до активів у володінні;
– продаж та обмін активів.

4.1.1.2. Для адміністратора системи:

– додавання та створення нових активів;
– контроль емісії.

4.1.2. Розробку виконати на блокчейн мережі Ethereum, мова програмування NodeJS.

4.1.3. Додаткові вимоги:

Реалізувати логіку обміну та покупки на рівні смарт контрактів.

4.2. Вимоги до надійності

4.2.1. Передбачити контроль введення інформації.

4.2.2. Передбачити захист від некоректних дій користувача.

4.2.3. Забезпечити цілісність інформації в базі даних.

4.2.4. Реалізувати архітектуру смарт контрактів таким чином, щоб дані про операції користувачів та володіння активами були відновлюваними у випадку бази даних.

4.3. Умови експлуатації

4.3.1. Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2. Обслуговування

4.3.3. Обслуговуючий персонал має складатися з одного адміністратора з доступом власника смарт контрактів. Інших персонал не потрібен через децентралізовану модель додатка.

4.4. Вимоги до складу і параметрів технічних засобів

4.4.1. Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2. Мінімальна конфігурація технічних засобів:

4.4.2.1. Тип процесору: AMD EPYC 7501 (1 ядро).

4.4.2.2. Об‘єм ОЗП: 512 Мб.

4.5. Вимоги до інформаційної та програмної сумісності

4.5.1. Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Unix.

4.5.2. Формати вхідних даних.

4.5.3. Програмне забезпечення повинно використовувати стандартизовані смарт контракти токенів для того, щоб інші сервіси також могли використовувати створені смарт контракти.

4.6. Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7. Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8. Спеціальні вимоги

Згенерувати інструкцію по розгортанню програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1. Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2. Програмне забезпечення повинно мати довідникову систему

5.3. У склад супроводжувальної документації повинні входити наступні документи:

5.3.1. Пояснювальна записка не менше ніж на 50 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2. Технічне завдання.

5.3.3. Керівництво користувача.

5.3.4. Програма та методика тестування.

5.4. Графічна частина повинна бути виконана на аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1. Схема структурна варіантів використання.

5.4.1. Схема структурна бізнес процесів.

5.4.3. Креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	19.03.2020	
2.	Розробка технічного завдання	30.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	02.04.2020	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	13.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	29.04.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	04.05.2020	Тести, результати тестування

7.	Розробка матеріалів текстової частини проєкту	11.05.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проєкту	13.05.2020	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	15.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1. Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
14.06.2020 01:15:58 EEST

Дата звіту:
14.06.2020 21:56:54 EEST

ID перевірки:
1004022444

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Krolevetskyi_ip61

ID файлу: 1004035523 **Кількість сторінок:** 46 **Кількість слів:** 6090 **Кількість символів:** 45984 **Розмір файлу:** 4.78 MB

9.8% Схожість

Найбільша схожість: 7.04% з джерело бібліотеки. **ID файлу:** 1000037963

7.52% Схожість з Інтернет джерелами	22	Page 48
9.7% Текстові збіги по Бібліотеці акаунту	75	Page 48

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Не знайдено замінених символів

Факультет інформатики та обчислювальної техніки Кафедра автоматизованих систем обробки інформації і управління

"ЗАТВЕРДЖЕНО"

В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*
(підпис) (ім'я, прізвище)

"___" _____ 2020 р.

Веб-платформа з продажу та обміну токенізованих активів

Програма та методика тестування

КПІ.ІП-6113.045440.04.51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Д.Ю. Кролевецький

Київ – 2020 року

ЗМСТ

1 ОБ'ЄКТ ВИПРОБУВАННЯ

Веб-платформа для продажу та обміну токенізованими, яка являє собою програмний продукт, що складається зі смарт контрактів у мережі Ethereum, серверної частини, розробленої з використання мови програмування NodeJS, бази даних Postgres та клієнтської web частини.

2 МЕТА ТЕСТУВАННЯ

Під час тестування в першу чергу слід звернути увагу на наступне:

– належний рівень безпеки даних;

– відповідність дизайну до вимог технічного дизайну;

– правильна робота елементів інтерфейсу;

– інтуїтивність користувацького інтерфейсу;

– коректність результатів роботи додатка.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування обраний метод Gray Box Testing. Перевіряється код та безпосередньо сам додаток на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

– тестування продуктивності;

– функціональне тестування;

– тестування стабільності;

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВВАННЯ

Працездатність перевіряється наступним чином:

– динамічне ручне тестування;

– тестування на відповідність продукту функціональним вимогам;

– статичне тестування коду;

– тестування елементів інтерфейсу.

Таблиця 4.1 – Відношення впливу до якості

Характеристика впливу / Характеристика Росту якості		Функціональність		Надійність	
		Придатність	Захищеність	Відмовостійкість	Завершеність
Функціональність	Придатність	+	-	-	+
	Захищеність	-	+	-	+
Надійність	Відмовостійкість	-	-	+	-
	Завершеність	+	-	-	+
Зручність використання	зрозумілість	+	+	+	-
	освоюваність	+	+	-	-
Супроводжуваність	гнучкість	+	-	+	-
	можливість тестування	-	-	-	+

Таблиця 4.2 - Вагові атрибути

Характеристика	**Атрибут**	**Вага**
Функціональність	Придатність	0.7
	Захищеність	1
Надійність	Відмовостійкість	0.8
	Завершеність	0.7
Зручність використання	зрозумілість	0.5
	освоюваність	0.3
Супроводжуваність	гнучкість	1
	можливість тестування	0.8

Таблиця 4.3 - Відображення моделі зовнішньої якості на якість

Характеристика впливу / Характеристика Росту якості		Експлуатаційна якість			
		Продуктивність	Задоволеність	Результативність	Безпечність
Функціональність	Придатність	+	-	-	+
	Захищеність	-	+	-	+
Надійність	Відмовостійкість	-	-	+	-
	Завершеність	+	-	-	+

Продовження таблиці 3.3

Зручність використання	Зрозумілість	+	+	+	-
	Освоюваність	+	+	-	-
Супроводжуваність	Гнучкість	+	-	+	-
	Можливість тестування	-	-	-	+

Таблиця 4.4 – Характеристики - вага

Характеристика	**Вага**
Результативність	0.4
Продуктивність	0.5
Безпечність	0.6
Задоволеність	0.6

Факультет інформатики та обчислювальної техніки Кафедра автоматизованих систем обробки інформації і управління

"ЗАТВЕРДЖЕНО"

В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*
(підпис) (ім'я, прізвище)

"___" _____ 2020 р.

Веб-платформа з продажу та обміну токенізованих активів

Керівництво користувача

КПІ.ІП-6113.045440.05.34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ Д.Ю. Кролевецький

Київ – 2020 року

ЗМСТ

1 ОСНОВНИЙ ІНТЕРФЕЙС

Для роботи з програмним забезпеченням необхідно встановити застосунок MetaMask [5]. Цей додаток дозволить підключитись до мережі Ethereum, та створити власний гаманець. Використовуватись даний додаток буде переважно для підпису транзакцій.

Після запуску клієнт-серверної частини додатка переходимо на http://localhost:4010/. Основний каталог додатку можемо бачити на рисунку 1.

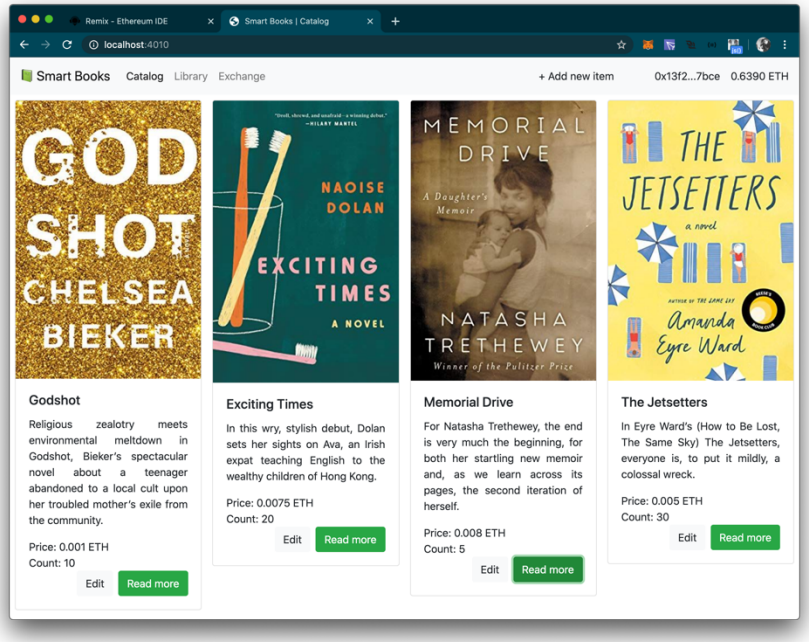


Рисунок 1 – Вигляд основного каталогу

2 ОПЕРАЦІЯ КУПІВЛІ

На сторінці каталогу користувач обирає книгу, яку хоче придбати. Для цього треба натиснути “Read more” на обраному блоці з зображенням книги. Після цього відкривається модальне вікно з додатковою інформацією – рисунок 2.

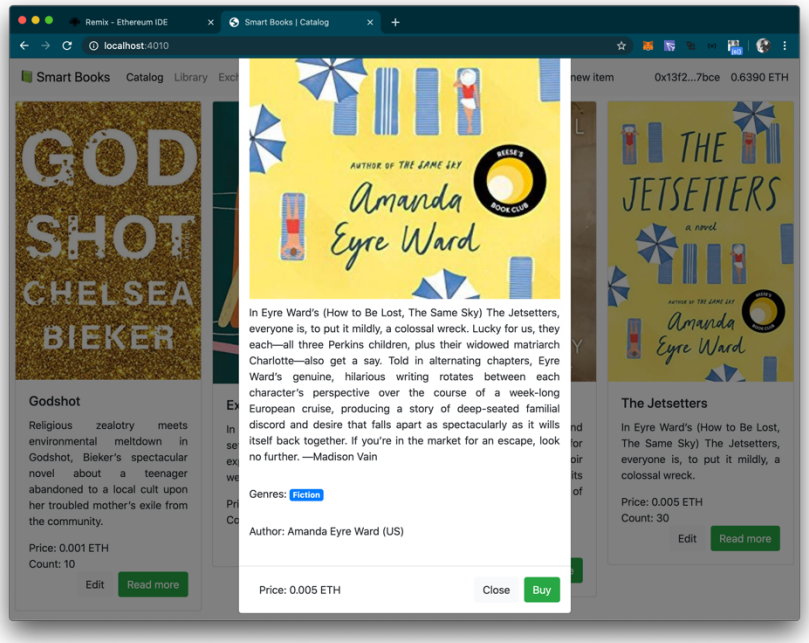


Рисунок 2 – Додаткова інформація про книгу

Наступним кроком користувач купує книгу. Для цього необхідно натиснути кнопку “Buy” та підписати транзакцію покупки – рисунок 3.

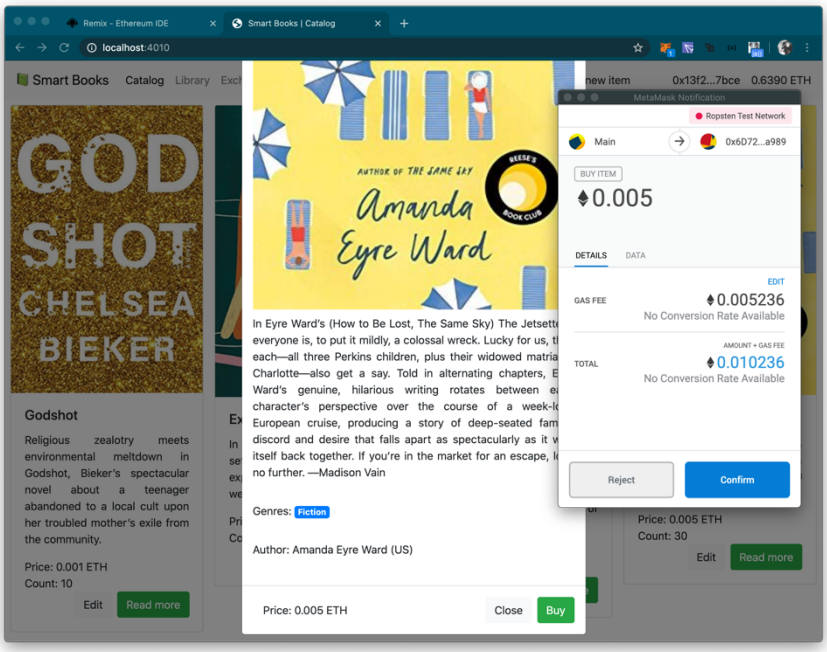


Рисунок 3 – Операція покупки книги

Після підписання транзакції та її підтвердження зі сторони мережі відповідна книга з'явиться у особистій бібліотеці. Посилання знаходиться у навігаційному барі. Вигляд особистої бібліотеки на рисунку 4.

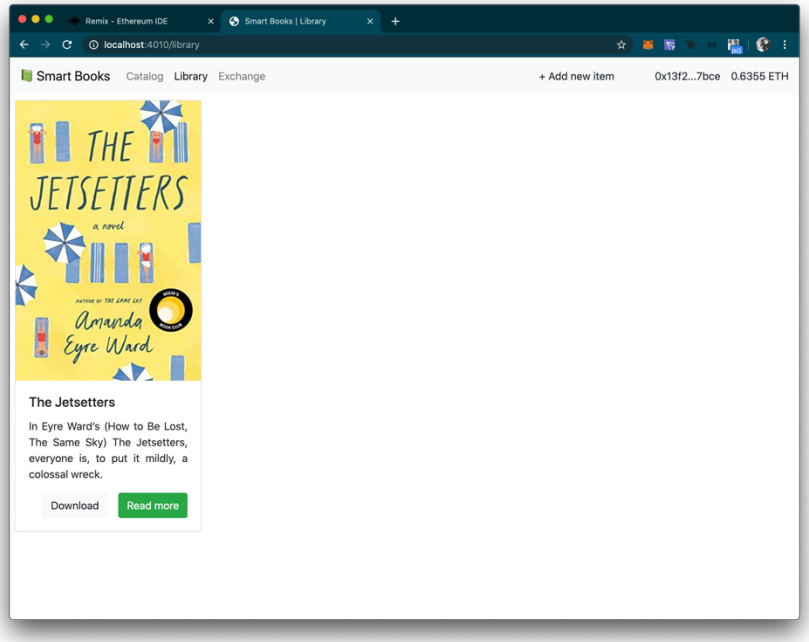


Рисунок 4 – Особиста бібліотека

3 ОПЕРАЦІЯ ОБМІНУ

Розглянемо механіку роботи обміну книгами. Переходимо на сторінку “Exchange” (посилання знаходиться у верхньому навігаційному барі). Вигляд сторінки обміну представлений на рисунку 5. У верхній частині представвлені власні запити на обмін. У нижній доступні запити від інших користувачів.

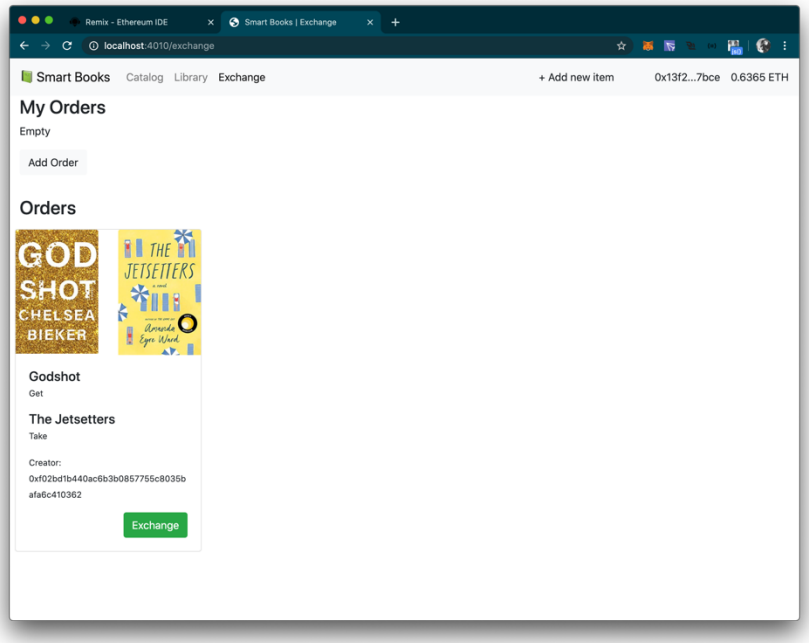


Рисунок 5 – Сторінка обміну книгами

Наступним кроком виконаємо запит на обмін. Для цього натискаємо кнопку “Exchange” біля доступної пропозиції. Після цього необхідно підписати дві транзакції – одна для того, щоб смарт контракт обмінника отримав доступ до токена користувача, а інша для створення власне запиту. Дивись рисунок 6.

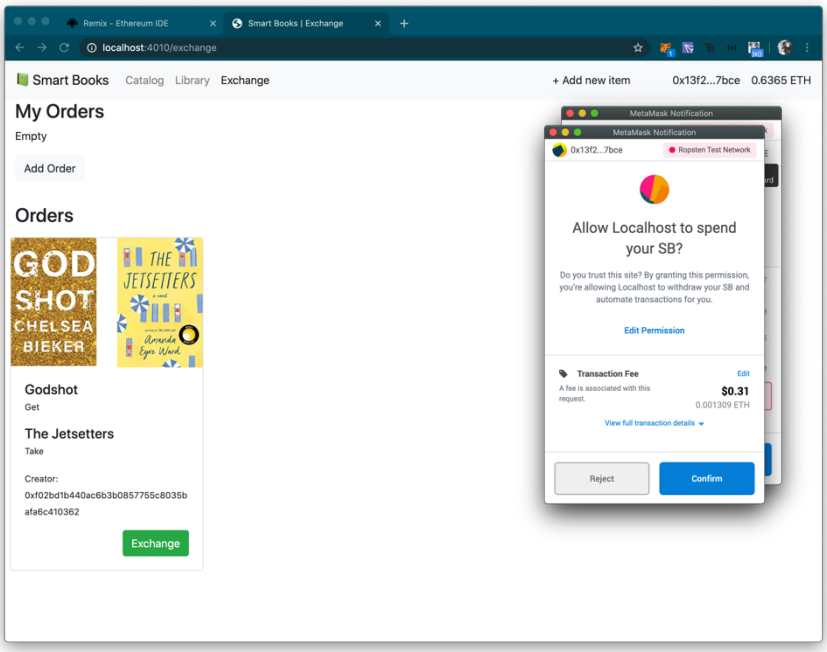


Рисунок 6 – Виконання операції обміну

Після підписання транзакцій бажана книга з’явиться у особистій бібліотеці, а віддана – зникне (рисунок 7).

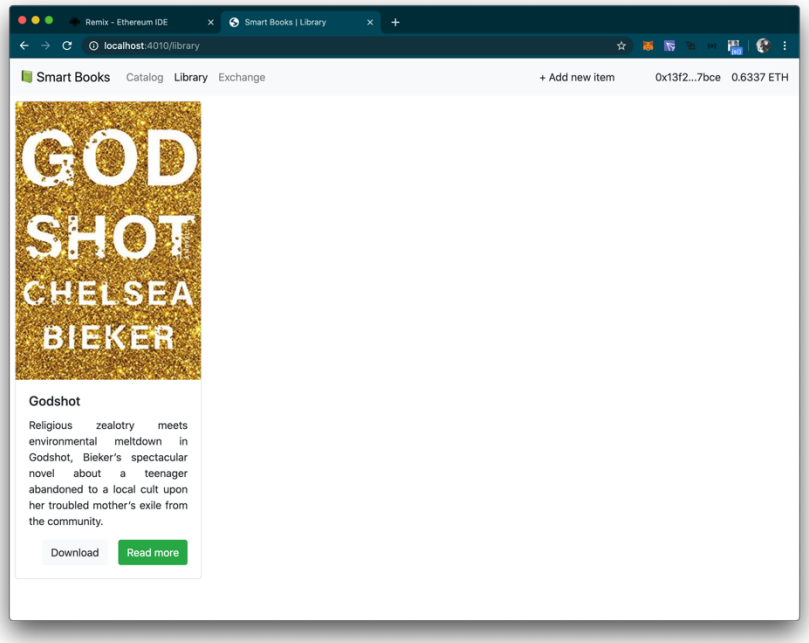


Рисунок 7 – Результат обміну

Останнім кроком продемонструємо створення власного запиту. Для цього на сторінці обміну необхідно натиснути “Add Order” та заповнити форму. Далі підписуємо знову дві транзакції – рисунок 7.

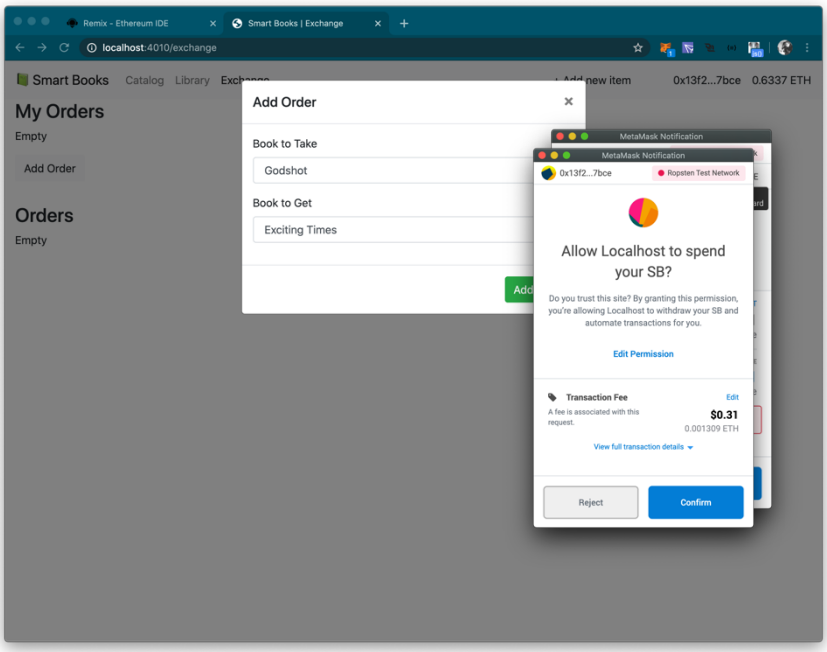


Рисунок 8 – Створення запиту на обмін

Після створення запиту він з’явиться у верхній частині сторінки запитів на обмін та буде доступний усім користувачам (рисунок 9).

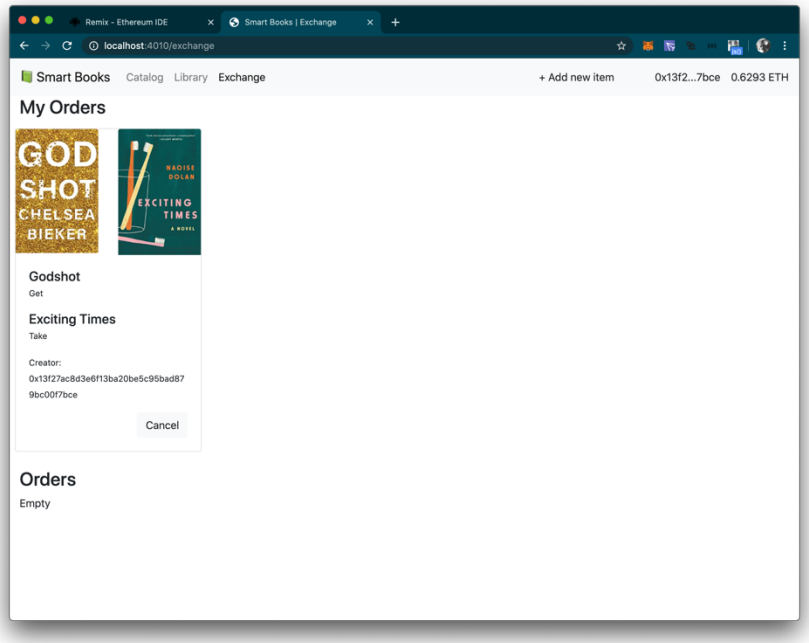


Рисунок 9 – Власний запит на обмін

Факультет інформатики та обчислювальної техніки Кафедра автоматизованих систем обробки інформації і управління

"ЗАТВЕРДЖЕНО"

В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)

"___" _____ 2020 р.

Веб-платформа з продажу та обміну токенізованих активів

Графічні матеріали

КПІ.ІП-6113.045440.06.99

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Ю.О. Олійник

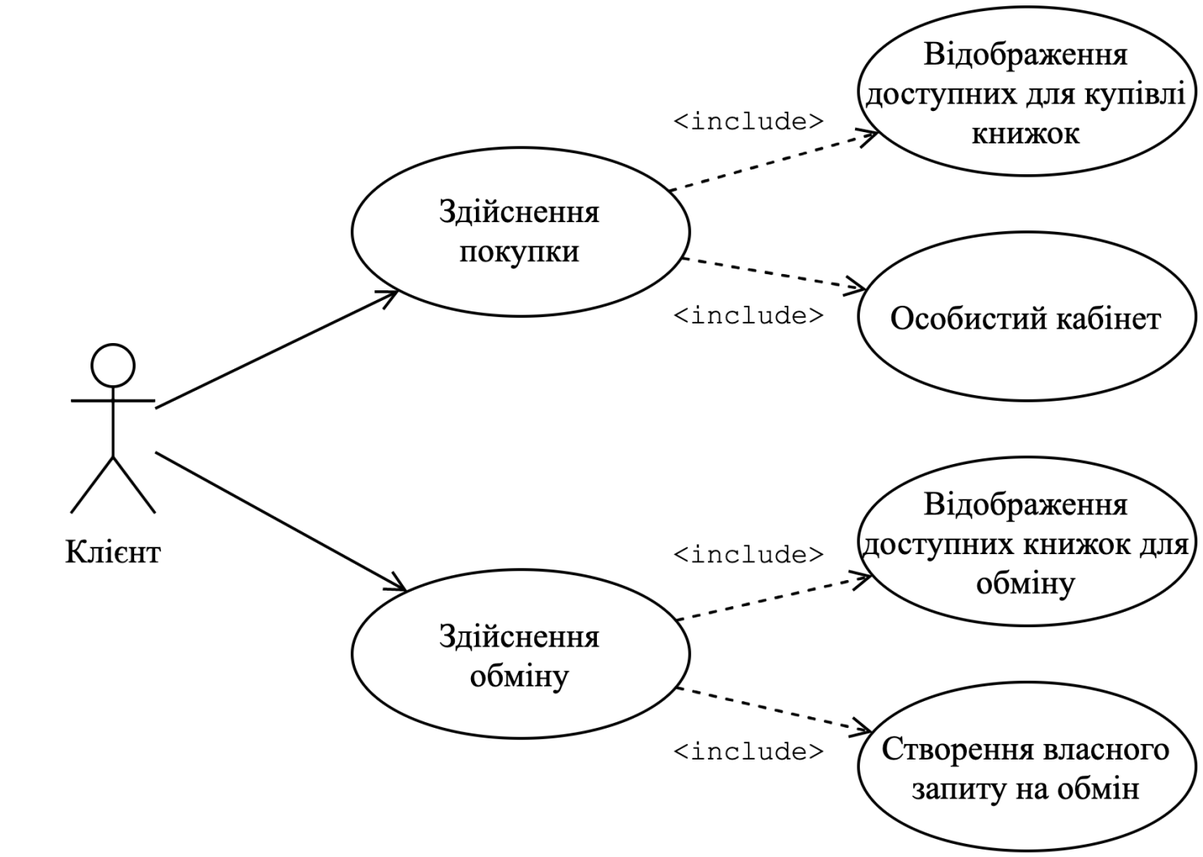
Нормоконтроль:

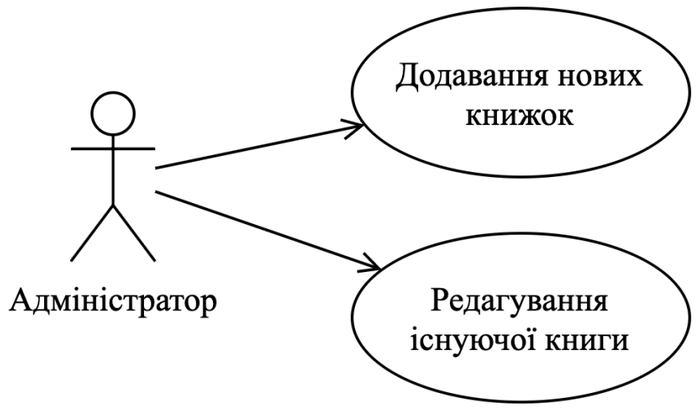
_____ К.І. Ліщук

Виконавець:

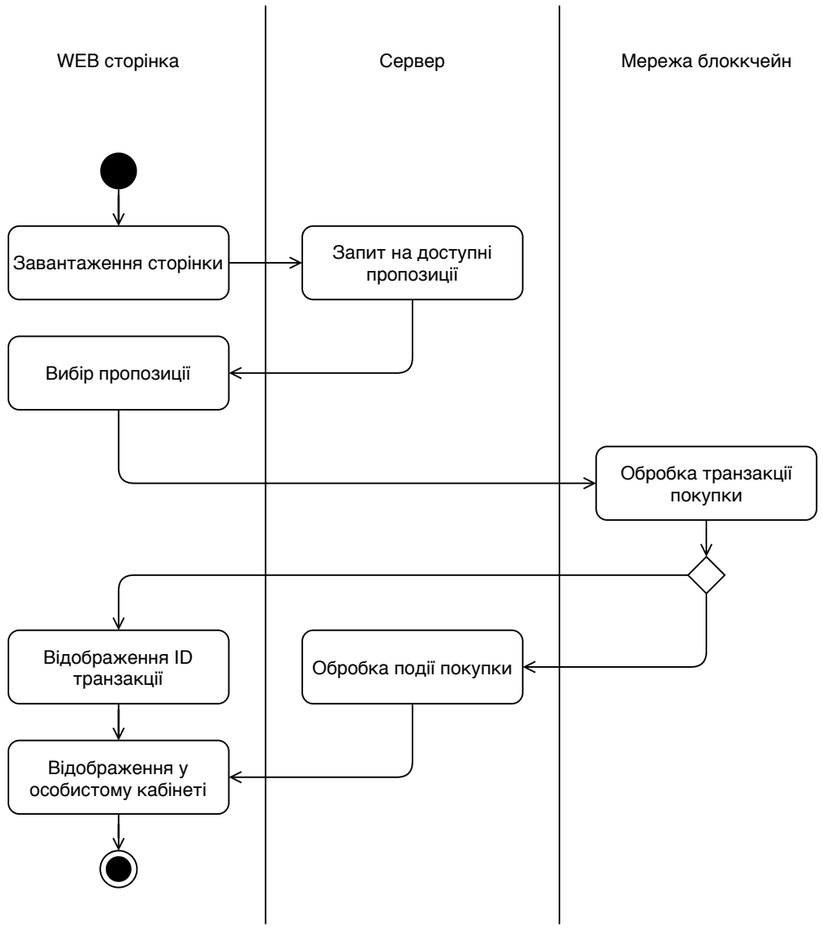
_____ Д.Ю. Кролевецький

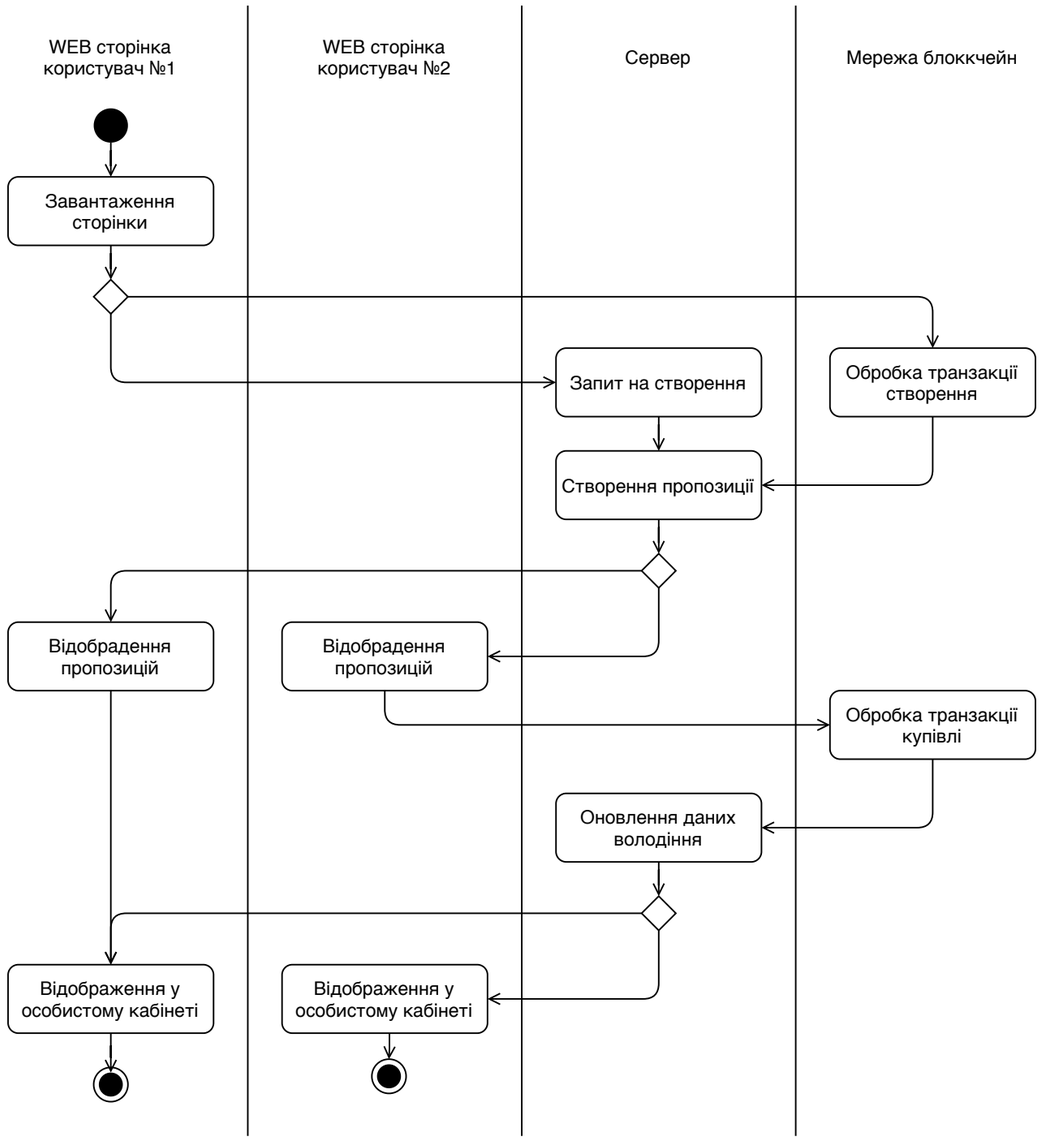
Київ – 2020 року





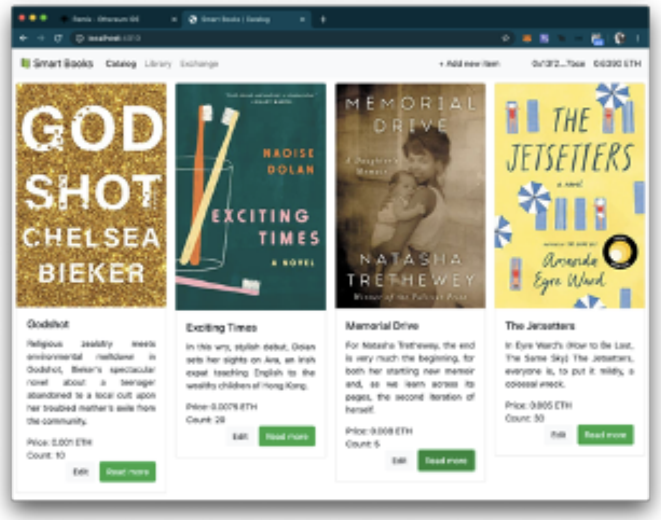
					КПІ.ІП-6113.045440.06.99.СС			
					Схема структурна варіантів використання	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Кролевецький Д.Ю.				Аркуш 1	Аркушів 1	
Перевірив		Олійник Ю.О.						
Т. кон.								
					Веб-платформа з продажу та обміну токенізованих активів	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61		
Н. кон.		Ліщук К.І.						
Затвердив		Олійник Ю.О.						



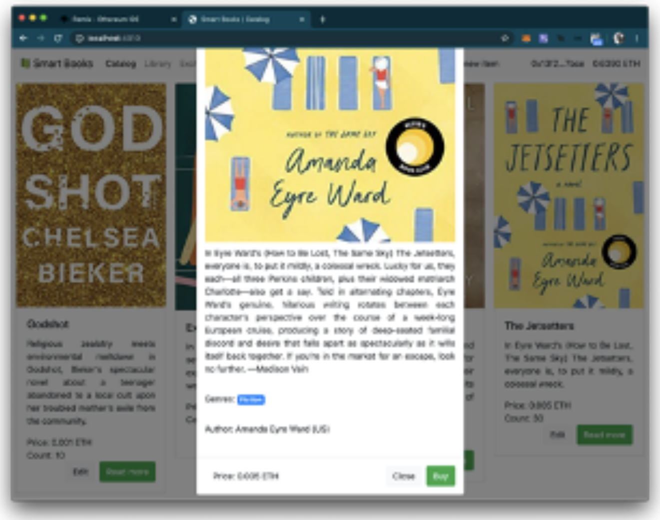


					КПІ.ІП-6113.045440.06.99.СС		
					Схема структурна бізнес процесів	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Кролевецький Д.Ю.				Аркуш 1	Аркушів 1
Перевірив		Олійник Ю.О.					
Т. кон.							
Н. кон.		Ліщук К.І.			Веб-платформа з продажу та обміну токенізованих активів	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61	
Затвердив		Олійник Ю.О.					

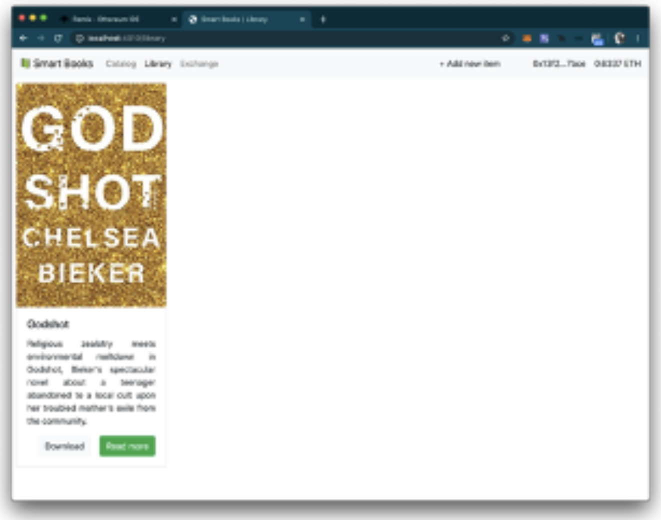
КПІ.ІП-6113.045440.06.99.КЕ



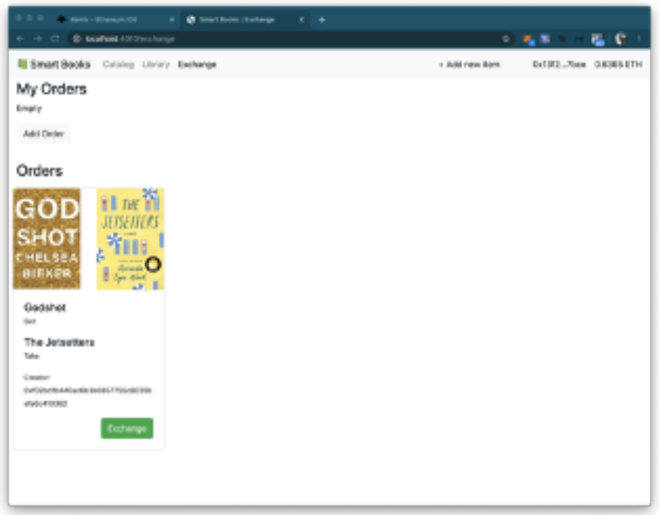
ЕКРАННА ФОРМА КАТАЛОГУ



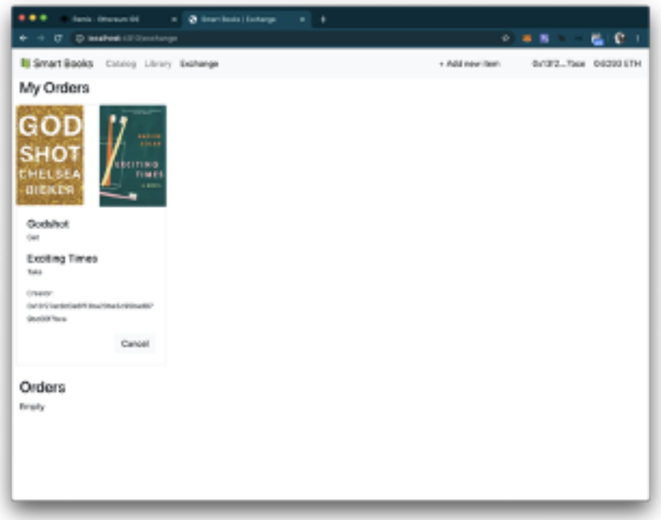
ЕКРАННА ФОРМА ОГЛЯДУ КНИГИ



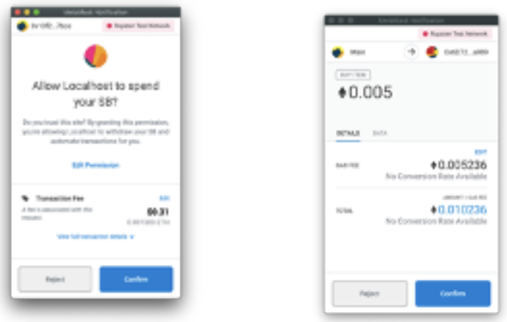
ЕКРАННА ФОРМА БІБЛІОТЕКИ КОРИСТУВАЧА



ЕКРАННА ФОРМА СТОРІНКИ ОБМІНУ



ЕКРАННА ФОРМА СТОРІНКИ ОБМІНУ З ВЛАСНОЮ ПРОПОЗИЦІЄЮ



ЕКРАННА ФОРМА ПОВІДОМЛЕНЬ METAMASK

					КПІ.ІП-6113.045440.06.99.КЕ			
Зм.	Арк.	№ документа	Підпис	Дата		Літера	Маса	Масштаб
Розробив		Кролевецький Д.Ю.			Креслення екранних форм			
Перевірив		Олійник Ю.О.						
Т. кон.						Аркуш 1		Аркушів 1
					Веб-платформа з продажу та обміну токенізованих активів	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-61		
Н. кон.		Ліщук К.І.						
Затвердив		Олійник Ю.О.						